Universitat de Lleida
Escola Politècnica Superior

Development of the videogame

for Android devices

Title: Development of the videogame PONG for Android devices

Author: Marc Albert Tarrés Navarra

Directors: Montserrat Sendín and Cèsar Fernández

Grade: Computer Engineering

Summary

The current use of mobile devices has opened new trends of communication and connection between different devices. Since this provision was intended to study the market new technologies for connecting these devices, particularly through new technology *Wifi Direct*. Specifically, this project has developed the classic Pong game for Android devices with the ability to play using *Wifi Direct*. This technology could connect two or more devices and play with each other without the need for a server. This technology, though very recent, is expected to be useful in the near future since it ignores access points or intermediate software and all devices are connected to each other. The project has focused mainly on the technology used but also needed data and game features, design and future possibilities. Thanks to the endless possibilities they provide new mobile devices has been possible to achieve the objectives set for this project.

Abstract

The current use of mobile devices has opened new trends of communication and connection between different devices. From this provision of the market has wanted to study new technologies for connecting these devices, particularly through new technology Wifi Direct. Specifically, this project will develop the classic Pong game for Android devices as well as the opportunity to play with WiFi Direct and an external server. Mainly focuses on the connection of two smartphones by named technology, despite being very recent are expected to be useful in the near future. The project mainly focuses on the technology used, but also accurate data as the game features and future possibilities. Thanks to the endless possibilities they provide new mobile devices has been possible to achieve the objectives set for this project.

Index

Chapter 1. Introduction..... 9

Chapter 1.1. Context..... 9

Chapter 1.2. Motivations..... 9

Chapter 1.3. Objectives of work..... 10

Chapter 1.4. Document structure..... 11

Chapter 2. Context of the application developed..... 13

Chapter 2.1. Knowledge..... 13

Chapter 2.1.1. The first videogames..... 13

Chapter 2.1.2. Types of videogames..... 14

Chapter 2.1.3. Development concepts..... 15

Chapter 2.2. Evolution of the *Pong*..... 16

Chapter 2.2.1. *Pong Classic*..... 16

Chapter 2.2.2. *Glow Pong*..... 17

Chapter 2.2.3. *Ping Pong Matrix*..... 17

Chapter 2.2.4. *Ping Pong HD*..... 19

Chapter 2.2.5. *Pong Direct*..... 20

Chapter 3. Feasibility and project planning..... 22

Chapter 3.1. Identification of risks..... 22

Chapter 3.1.1. Technical risks..... 22

Chapter 3.1.2. Project risks..... 22

Chapter 3.1.3. Business risks..... 22

Chapter 3.2. Screening risks..... 23

Chapter 3.3. Feasibility study..... 24

Chapter 3.4. Planning..... 25

Chapter 3.5. Budget..... 26

Chapter 4. Technologies used and state of the art..... 28

Chapter 4.1. Platform used..... 28

Chapter 4.1.1. Platform *Android*..... 28

Chapter 4.1.1.1. Architecture..... 29

Chapter 4.1.1.2. Structure of an application..... 31
Chapter 4.1.1.3. Versions..... 34
Chapter 4.2. *Wifi Direct*..... 37
Chapter 4.2.1. Technology overview..... 37
Chapter 4.2.2. Key features..... 38
Chapter 4.2.3. Operation..... 40
Chapter 4.2.4. *Wifi Direct* devices..... 42
Chapter 4.3. Runtime environment..... 43
Chapter 4.3.1. *Java Virtual Machine*..... 43
Chapter 4.3.2. *Dalvik*..... 44
Chapter 4.4. Development tools..... 46
Chapter 4.4.1. *Eclipse*..... 46

Chapter 5. Requirements analysis..... 49
Chapter 5.1. Functional requirements..... 49
Chapter 5.2. Non-functional requirements..... 50
Chapter 5.3. Use case diagram..... 51
Chapter 5.3.1. Diagram play against the device..... 51
Chapter 5.3.2. Diagram play using *Wifi Direct*..... 51
Chapter 5.3.3. Diagram played with a server..... 52
Chapter 5.4. Specification of use cases..... 53
Chapter 5.5. System specification..... 57
Chapter 5.5.1. Hardware resources..... 57
Chapter 5.5.2. Technical restrictions..... 58

Chapter 6. Interface design..... 59
Chapter 6.1. Design screens and menus..... 64
Chapter 6.2. Design menus..... 66
Chapter 6.3. Initial screen..... 68
Chapter 6.4. Game screen..... 71
Chapter 6.5. *Wifi Direct* screen connection..... 71

Chapter 7. Implementation..... 73
Chapter 7.1. Application development..... 73
Chapter 7.1.1. *Pong*..... 73

Chapter 7.1.1.1. *Handler*..... 73
Chapter 7.1.1.2. Artificial intelligence..... 75
Chapter 7.1.1.3. *Shared Preferences*..... 77
Chapter 7.1.1.4. *PreferenceActivity*..... 80
Chapter 7.1.2. *Wifi Direct*..... 82
Chapter 7.1.2.1. *BroadcastReciver*..... 82
Chapter 7.1.2.2. *WifiManager*..... 84
Chapter 7.1.2.3. *Comparable* interface..... 86
Chapter 7.1.2.4. *TimerTask*..... 87
Chapter 7.1.2.5. *Socket*..... 88

Chapter 8. Conclusions and future work..... 94
Chapter 8.1. Future work..... 94

Bibliography..... 96
Books..... 96
Web pages..... 96

List of tables and figures

Figure 1. Pong Classic main screens..... 16
Figure 2. Glow Pong main screens..... 17
Figure 3. Ping Pong Matrix main screens..... 18
Figure 4. Ping Pong HD main screens..... 19
Figure 5. Statistics Ping Pong HD..... 20
Figure 6. Pong Direct main screens..... 21
Figure 7. Project planning..... 26
Figure 8. Android architecture [22] 29
Figure 9. *Androd* lifecycle [23] 32
Figure 10. *Android* versions [24] 34
Figure 11. *Wifi Direct* on Android..... 37
Figure 12. Connection *Wifi Direct* devices [25] 39
Figure 13. Classic home network [26] 40
Figure 14. Network with *Wifi Direct* [27] 41
Figure 15. JVM scheme [28] 44
Figure 16. *Dalvik* running scheme [29] 45
Figure 17. Screenshot Eclipse program..... 48
Figure 18. Use case diagram: Play against the device..... 51
Figure 19. Use case diagram: Play through *Wifi Direct*..... 51
Figure 20. Use case diagram: Play through a server..... 52
Figure 21. Bitmap example [31]..... 61
Figure 22. Various bitmaps [32] 61
Figure 23. Bitmaps classification [33] 62
Figure 24. Fragments [34] 62
Figure 25. Android Patterns [35] 63
Figure 26. Menu code fragment..... 64
Figure 27. Menu creation code fragment..... 66
Figure 28. Initial screen..... 66
Figure 29. Canvas code fragment..... 67
Figure 30. Examples of use canvas..... 68
Figure 31. Examples of use paint..... 69
Figure 32. Game screens..... 69

Figure 33. Create new game screen..... 70
Figure 34. Play into existing game..... 71
Figure 35. Error before use Handler..... 73
Figure 36. Handler fragment to update the screen..... 74
Figure 37. Handler fragment to notify..... 74
Figure 38. Handler fragment to update..... 74
Figure 39. Switch fragment..... 75
Figure 40. Fragment IA exact code..... 75
Figure 41. Fragment following ball IA code..... 75
Figure 42. Fragment IA code..... 76
Figure 43. Fragment user preferences..... 78
Figure 44. Fragment mute game..... 79
Figure 45. Fragment XML code..... 79
Figure 46. Fragment XML *PreferenceActivity* code..... 80
Figure 47. *BroadcastReceiver* schema [36] 82
Figure 48. Fragment *BroadcastReceiver* code..... 83
Figure 49. Fragment *BroadcastReceiver* code..... 84
Figure 50. Fragment *WifiReceiver* code..... 85
Figure 51. Fragment *WifiReceiver* code..... 85
Figure 52. Fragment *Comparable* code..... 87
Figure 53. Fragment *TimerTask* code..... 88
Figure 54. Sockets connection diagram..... 89
Figure 55. Fragment sockets code..... 91
Figure 56. Fragment client code..... 92

Table 1. Table risks..... 23
Table 2. Measures to minimize the impact..... 23
Table 3. DAFO analysis..... 24
Table 4. Project budget..... 27
Table 5. Screens densities on Android [30] 60

1. Introduction

1.1. Context

Android videogames are gaining importance in the universe of the Google's Operative system addressed for mobile phones. The quality of them is getting better every time, not only for their technical finish but also for their development. Within this tendency, one of the most asked for videogames are the multiplayer because they let users compete with each other anywhere. This evolution was used when creating the application that will be described during this project, which is based in new technologies and platforms for the development of multiplayer videogames.

1.2. Motivations

It mainly exist two motivations as the answer to the idea of developing this project In the first place, my own interest for the world of videogames and all which surrounds it. Since I was young I have always been interested in classical videogames (also called Vintage) as for instance the first 2D videogames or even the classics like PONG. I have experienced a huge variety of technologic toys ever since I owned my first Super Nintendo NES. After having played a great amount of different types of videogames I can say I do prefer the classical ones nowadays.

My other motivation is to overcome my personal boundaries knowing and developing a project using relatively modern technology that is used by millions. After having studied terms as Cloud and subjects related to mobile phones' dispositive programming, I wanted to go further and develop an application applying everything learned in advance as well as discovering new APIs and learning new programming methods.

1.3. Project's objectives

The following project is based in the design and implementation of a videogame for mobile phones, concretely for the Android operative system. This system is characterized by being an open code system, which lets anybody to modify and improve it. This means that behind the platform there is a huge community that can provide support when developing new

applications for this platform. What is more, the programming in this platform is based in Java code which is well-documented in books or articles on the internet.
On the other hand, the Android system has a great portability and flexibility, which gives simplicity when migrating from an application for mobile phones to an application for digital tablets. It is important to talk about the great amount of Application Programming Interfaces (APIs), which can be freely used and enhance the programming of such important part as the analysis of sensors or the usage of Bluetooth and Wi-Fi connections.

The application that will be developed is based in one of the first videogames that appeared more than thirty years ago. More precisely, the videogame Pong. This videogame was characterized by being similar to the Tennis videogame but without needing a net. Each player disposes of a tennis racket, which has to be used to hit the ball and avoid it to cross beyond the player's zone. If any of both players lets the ball cross, the other player will obtain one point. The winner will be the player who gets more points after 10 games.
Basing on this first objective, the development of the videogame for the Android system will be developed but focusing on the following videogame options:

- Player against SO: the player will have to beat the own game's algorithm.
- Player against player using Wi-Fi-direct: two different players will confront thanks to Wi-Fi-direct technology.
- Player against player using the server: two players will confront independently of their geographical location.

In order to complete the various game options will acquire the knowledge necessary to achieve the following objectives: The discovery and subsequent use of new technologies such as *WiFi Direct*, *Google App Engine* or *Node*:

- Developing a game in the Android platform
- Interaction and passing information between different mobile devices
- The design and modelling application from scratch
- The development of a mechanism capable of artificial intelligence to play against a real user
- The completion and launch of the application

To complete the application in question will solve a number of important issues, such as what data to send when playing two real players, which algorithm to use in order to play against the computer or even as implement the server into the record of the number of games played and won the average.

Game Pong was chosen as one of the most classic games and when simpler to implement, in order to set aside the importance of running the game and focus on the objective of the study and Android its features. This is the point where you wanted to increase the personal background knowledge of new forms of communication between devices and their operation. Furthermore, this work will provide significant learning because the information sought and all code developed will be made possible by research staff conducted.

1.4. Document structure

The following document is divided into three main chapters; context (chapters 1, 2, 3 and 4) development (chapters 5, 6 and 7) and conclusions (Chapter 8). The first chapter is done in-depth study of the different technologies and concepts directly related to the job. The second chapter focuses on the process followed when developing and implementing technical decisions and problems encountered during the implementation. The third chapter in the conclusions of the project shows the results obtained throughout development and are continuations of possible improvements and future work. Below is a view of the different chapters briefly:

- **Chapter 1:** In this first chapter focuses on the context of the application and the main objectives of the study and use of new communication technologies and the development of a video game in the Android platform.

- **Chapter 2:** It shows all the research accomplished during the project and the analysis carried out in a number of applications already on the market with features similar to carry out in this project.

- **Chapter 3:** This chapter identifies all potential risks of the application as well as the possibilities to overcome these risks. Moreover carried out a feasibility study of the project based on the information gathered with the help of a SWOT table.

- **Chapter 4:** One of the most important chapters in which the project is centred on the assumed knowledge related to the used platform (Android) and the technology to be used (*Wifi Direct*). This chapter shows the main features of both issues and how to use them.

- **Chapter 5:** The requirements analysis is the emphasis on different application requirements to ensure the proper functioning of this. Moreover carried out the study and analysis of the various use cases of the project.

- **Chapter 6:** The design of chapter focuses on how the end user will see the application, so called different patterns and design features of the game. Also explains the main design issues known in the Android platform.

- **Chapter 7:** Chapter deployment shows the main points of the inner workings of the application as well as the functions and parts necessary code for the proper functioning of the application.

- **Chapter 8:** The project conclusions are drawn in this section along with the options and future work from this project.

2. Contextualization of the application developed

2.1. Previous knowledge

2.1.1. The first videogames

One of the first videogames was the very well known *Nought and Crosses*, also known as OXO. It was developed by *Alexander S. Douglas* in 1952 [2]. This game was the first computerized version of what is known nowadays as *Three on a row*. Its main characteristic was that the player could play against the machine [1].

In 1958, William Higginbotham developed, thanks to a trajectories calculation program and an oscilloscope, a table tennis simulator. It was called *Tennis for two.* This game was the first one in which two people could play at the same time [2].

Four years later, Steve Russell spent more than six months in developing a game based on vector graphics: *Space War*. In this videogame, two players could take two spaceships under control and fight with each other.

Almost ten years later, Ralph Baer started to develop the videogame *Fox and Hounds* which was characterized by being one of the predecessors of the domestic console. It let the player play to some pre-set videogames in any television. But it was not until 1971 that the game *Computer Space* [3], started to be marketed. *Computer Space* was an upgraded version of *Space War*.

But, in fact, the expansion of videogames started with the gaming machine PONG (the commercial version of the videogame *Tennis for Two*). The system was designed and developed by ATARI. It introduced the machine in 1972. It was a very important step in the videogame industry. During the next few years, more videogames where implemented thanks to microprocessors and memory chips.

From that moment on, the evolution of videogames has been really quick. It has achieved domestic consoles, powerful graphic engines and a lot more. But, as this project is based on the videogame PONG, only the history that surrounds this game is important.

2.1.2. Types of videogames

A videogame is an electronic device, which allows, using the right commands, to simulate videogames on televisions and computers' screens or any other devices. In fact, a videogame is a software thought and developed for entertaining and interact with the computer or other players. This software works over an electronic dispositive. In this project, though, the focus will be on mobile phones.

Generally, videogames are grouped by their genre or purpose. Below there is a brief summary of each of the genres nowadays used [4]:

- **Action (arcade):** This is one of the most developed videogames and has its origin related to Arcade Games or gaming machines games. The term Arcade has been established for those games that challenge the player to complete different stages.

- **Strategy:** It is based on classical games with or without a board. Its objective is to command a great amount of characters, objects or data in order to achieve different objectives. There are two types of theme: management (like calculating costs) or fight (like controlling military forces).

- **Role:** Inspired on the classical games of role where the main character has to improve its abilities while interacting with the environment and other characters. They are usually characterized by the decisions of the character as they are directly connected to a near future. This means that, depending on their decisions, they will get good or bad results.

- **Adventure:** Another classic among the videogames world is this genre. It is characterized by guiding a character along the plot of a game. The character will have to interact with other characters and objects so that it can achieve the objectives of the game.

- **Sportive:** Based on sports as football or basketball, this genre converts the real world into a fictional one and the player can command great professionals of each sport.

- **Driving:** As the word says, it is focused on controlling vehicles (cars, motorbikes …) with the objective of winning races. Opponents are not always required.

- **Jigsaw:** This group includes all the games where an initial problem has to be solved achieving the intended goals. In this group, the games are mainly individual games and do not require an opponent.

Finally, it is important to say that many of the genres previously explained differ between the arcade videogames and the simulation ones. The first type has unreal location and objectives. The second type is faithful to reality.

2.1.3. Concepts of development

New terms and concepts have been slowly being introduced among the world of videogames. Below there is a brief summary of four of the most used and needed terms of this project [5]:

- **Sprites:** They are bit maps in 2D drawn directly in a representative destination without using the channeling of the transformation, lightning or effects. In other words, they are parts of the game that always maintain the same shape or image. They are usually used to show information as a state bar, the amount of lives left or the points. In some of the first games, sprites where the background as they were completely static. From the first sprites it was possible to create the first animations that were different sprites in motion.

- ***Framerate*:** In any videogame, it is very important that all the movements occur at the same speed, independently of the hardware behind. If this is not taken into account it is possible that, when installing a videogame in an ancient machine, it will run a lot slower than its original version. A *framerate* refers to the frequency in which the main cycle of the game is executed for a portion of time. The more quantity of FPS (frames per seconds), the best fluency the videogame will have. In videogames, the ranking of FPS goes between 60 and 100 FPS.

- **Game loop:** When a game is played it seems that everything is happening at the same time but, in fact, there are several times in which the processor can only process an instruction every time. The key is to do each task the fastest as possible and to do the next one before the user is able to see the next frame of the game, The game loop is the one in charge of deciding which task to be done at each moment, depending on the logging in of the user.

2.2. Evolution of the game *Pong*

Once the idea of the project has been settled, an analysis of five improved applications of the one chosen in this project has been done.

2.2.1. *Pong Classic*

- **Description:** It is Pong Classic but upgraded for the new platforms. It is an almost identic reinvention of the real game but with new functions.
- **Characteristics:**
    - Possibility of 1 or 2 players.
    - Easy and tactile controls.
    - Loyal to the original version (both source font and images)
- **Downloading:** https://play.google.com/store/apps/details?id=com.helldev.pongg
- **Screenshots:**

Figure 1. Pong Classic main screens

2.2.2. *Glow Pong*

- **Description:** Classic Pong updated with neon and particles.
- **Characteristics:**
    - Possibility of 1 or 2 players.
    - Easy and tactile controls.
    - Visually more modern (neon effect)
- **Downloading:** https://play.google.com/store/apps/details?id=com.xyz.pong
- **Screenshots:**

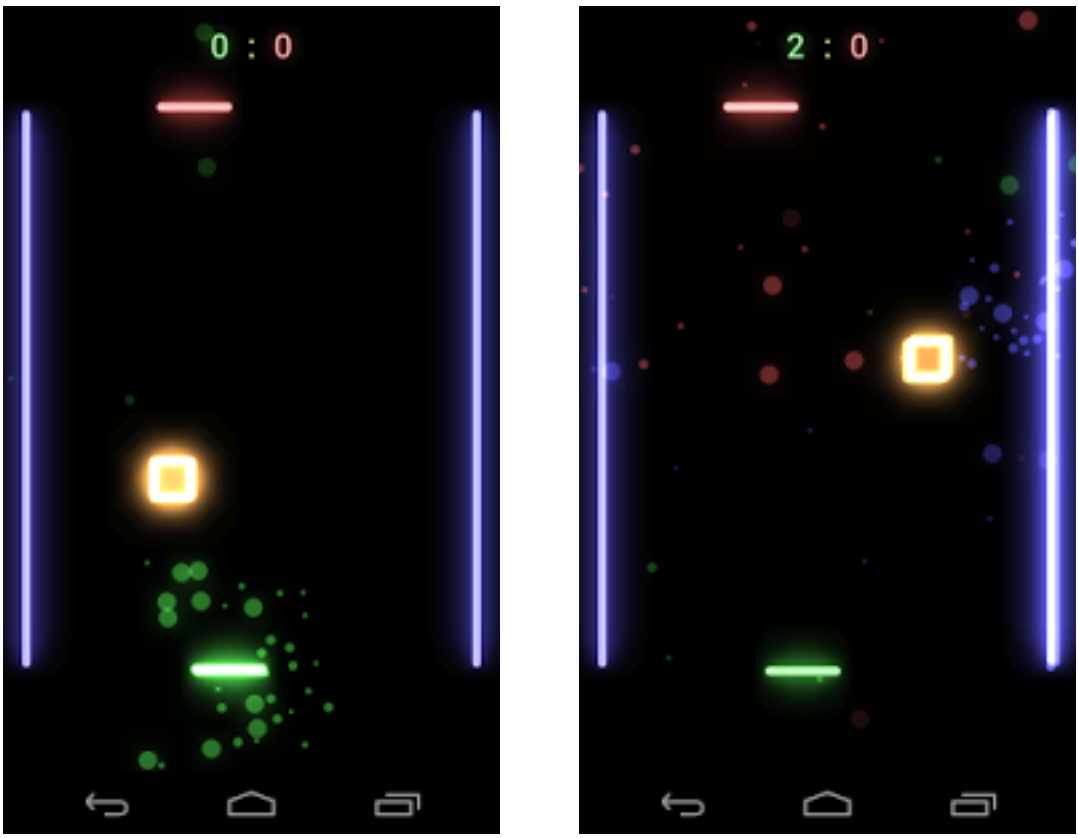

Figure 2. Glow Pong main screens

2.2.3. *Ping Pong Matrix*

- **Description:** Like the Classic Pong but only thought for two players. Each player has its pitch (the opponent's racket is not shown). Each player commands a racket and the ball travels between both devices (mobile phones).
- **Characteristics:**
    - Two players are needed as well as two devices with Bluetooth..

- Easy and tactile controls.
- Virtually classic.
- High power consumption because of the device sending data all the time.
- Limited only for devices with Bluetooth.

- **Downloading:**

  https://play.google.com/store/apps/details?id=com.androidesimple.pingpongmatrix

- **Screenshots:**

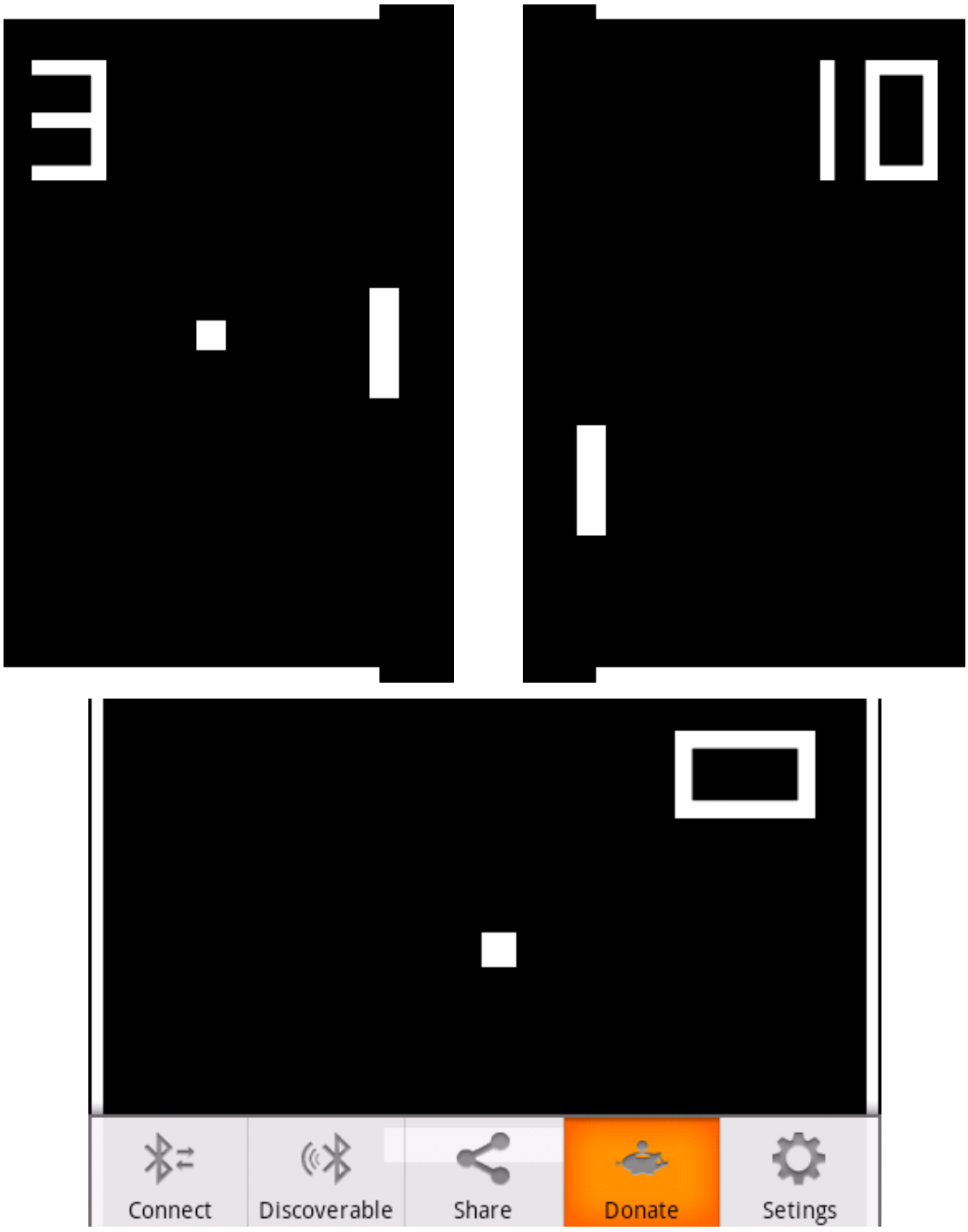

Figure 3. Ping Pong Matrix main screens

2.2.4. *Ping Pong HD*

- **Description:** Ping Pong HD is the most modern version of one of the most classic and popular game during the 70s. It is a game in 2D. The winner is the player with the highest score.
- **Characteristics:**
    - Possibility of 1 or 2 players.
    - Easy and tactile controls.
    - Visually more modern
    - 3 different levels of difficulty
    - Game statistics (won and lost games)
    - Retro sound effects
- **Downloading:** https://play.google.com/store/apps/details?id=com.cgames.pingpong

Screenshots:

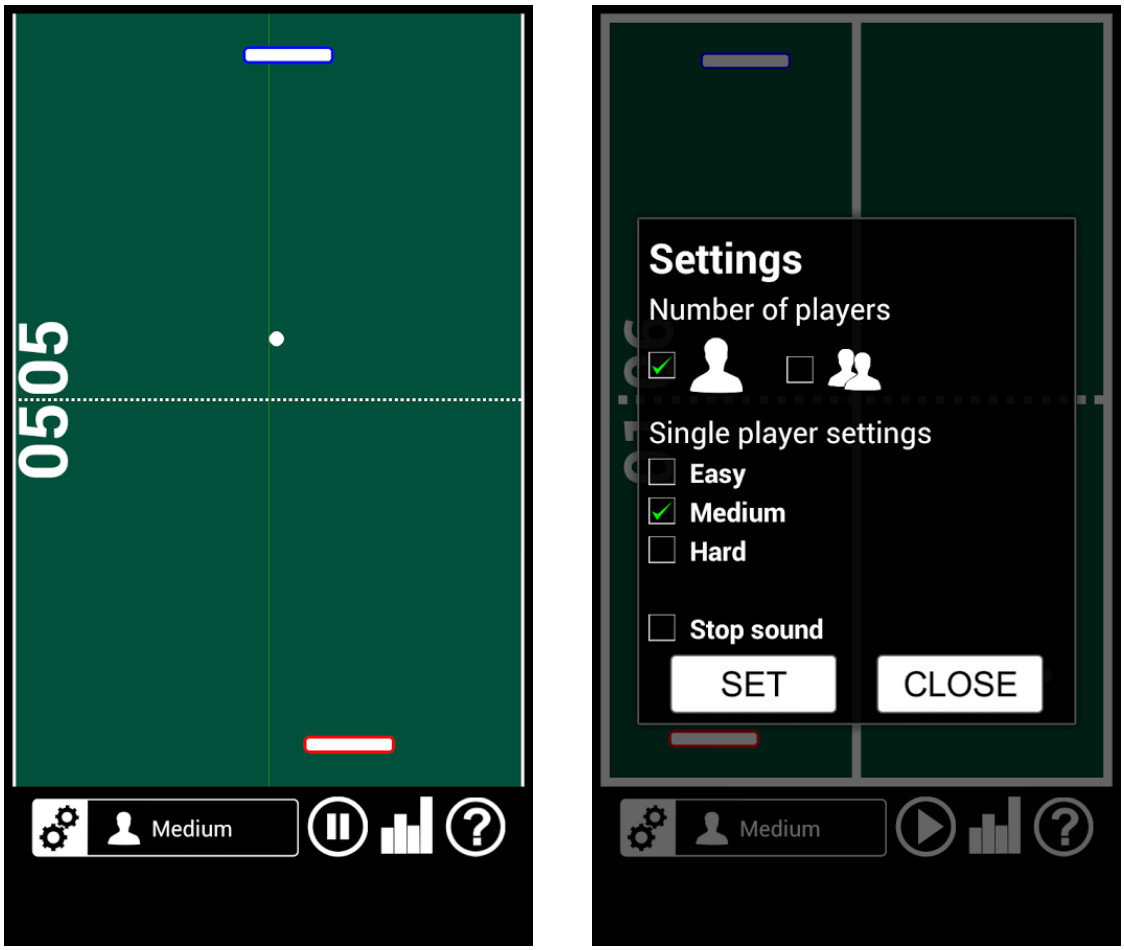

Figure 4. Ping Pong HD main screens

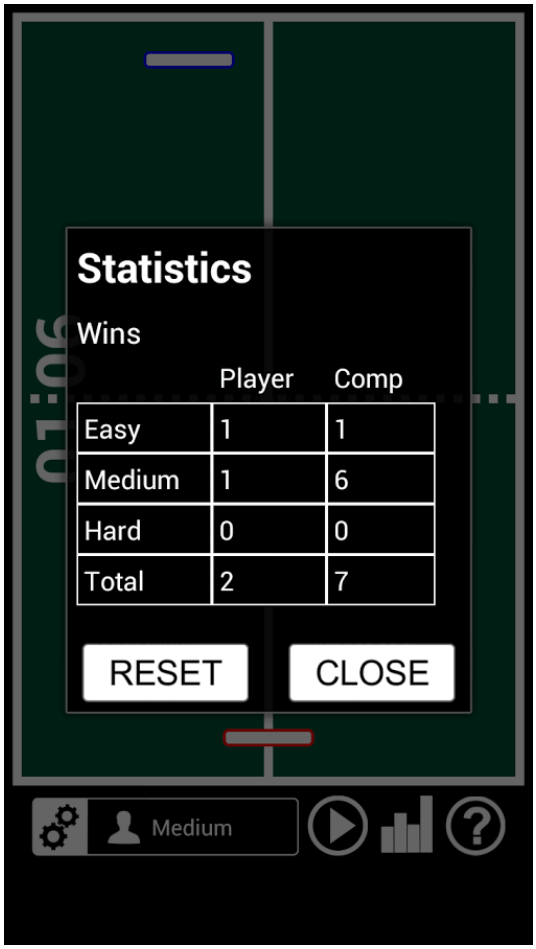

Figure 5. Statistics Ping Pong HD

2.2.5. *Pong Direct*

- **Description:** Inspired on the game PONG it is characterized for being able to be played by one or two players using Wi-Fi-Direct. Furthermore, the background can be chosen by the player to make it similar to a football or volleyball pitch and more.
- **Characteristics:**
  - Possibility of 1 or 2 players.
  - Easy and tactile controls.
  - Different options of visual style.
  - Wi-Fi connection
  - Retro effects and sounds.
  - Optimized for any technological device.
- **Downloading:** https://play.google.com/store/apps/details?id=fr.drouteam.pongdirect

Screenshots:

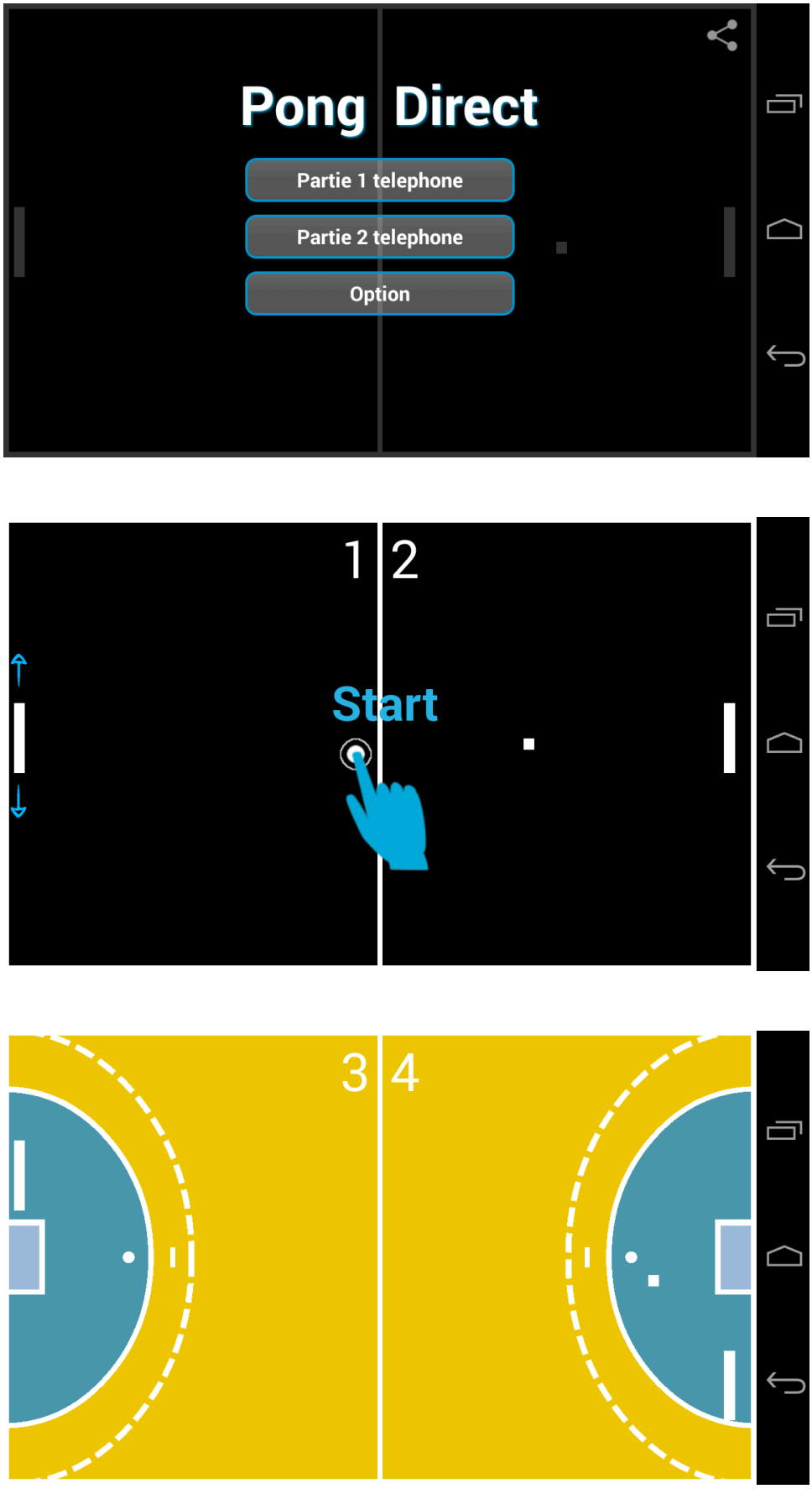

Figure 6. Pong Direct main screens

3. Feasibility and project planning

3.1. Risks identification

It is normal for all projects to have a certain grade of uncertainty. Some of these uncertainties or risks can become real problems along the project. They can also provoke not to achieve all the objectives asked (related to time, cost, quality…). Below there is a list of the different risks that can exist within this project. They are classified depending on the type of risk.

3.1.1. Technic risks

After intensive research and contrast of examples, I realized that the most difficult part is the one related with Wi-Fi-Direct technology. This technology is new and not developed in a great amount so there is a little risk when implementing the project as its complexity is not yet known.

3.1.2. Project risks

It exists a second type of risk: timing. The short term given to do the project and the huge amount of objectives to achieve are a dangerous mixture and can make it difficult to achieve the planning shown in section 3.3 of this project.

3.1.3. Trade risks

Selling this application can be affected by two different risks: the market and the budget. Depending on the problems that may appear during the whole project, the budget can change. This is why it is important to follow the planning and avoid any problem that could make the budget increase. At the same time, there is the market risk. This risk means that, after having launched the application, it can be unnoticed.

3.2. Risks projection

Once all the possible risks have been identified, I have done a risks chart, which shows the consequences of each risk and its impact:

Risk	Type of risk	Description	Probability	Impact
Wi-Fi-Direct technology	Technical	New technology short in examples	50%	1
Timing	Project	Insufficient time to finish the project	10%	2
Market	Trade	Little social acceptance	12%	3
Budget	Trade	Increase of the initial budget	25%	3

Table 1. Table risks

Impact categories:

1.Catastrophe, 2.Critical, 3.Marginal, 4.Negligible

Despite the potential risks, it also presents a table with the measures undertaken to minimize the impact:

Risk	Measures to minimize the impact
***Wifi Direct* tecnology**	Make use of the technology in other smaller projects or for information without real functionality. This will demonstrate how the study and use of this technology made an exhaustive study of the technology and implementing of the functionality of this technology.
Timing	Remove secondary objectives (delete avoid primary objectives). If some of the objectives are not met you can always remove the remaining objectives have enough weight in the project.
Market	Give a population study to ascertain the best way to enter the application market. Make use of video advertising channels and other applications so that users can download the application.
Budget	Limit the budget initially estimated cost of this project. If overcome this initial estimate by more than 20% extra will reflect on what goals to achieve to remove the maximum amount of the budget targets set initially.

Table 2. Measures to minimize the impact

3.3. Feasibility study

The SWOT (an acronym formed from the initials of the words Weaknesses, Threats, Strengths, Opportunities) or SWOT [6], the equivalent in English (or strengths Strengths, Weaknesses or weakness, opportunity or opportunities, or threats Threats) is an analytical tool initiator of strategic thinking, which allows us to make an introduction and mindset with which to perform a proper analysis of the competitive situation of a project. Thus, the method of SWOT analysis is to analyse the competitive environment of the business from both sides or environments: internal and external. Below is the SWOT analysis performed for this project:

Weaknesses	**Threats**
Technology poorly documented Few examples and without a community behind Without the gifts reputation PlayStore Little knowledge of technology Deadline limited	Technical limitations Technology with a lot of competition Alternatives in the framework of the implementation Competition with more resources and knowledge Little acceptance by users
Strengths	**Opportunities**
Classic application And many fans Known among the main user groups Small amount of data sent Similar versions of little known Opportunity to develop without major capital Motivation by Developer	Booming Technology Necessary in many applications Technology with future Chance of product diversification Then alters aspects of the project in the same game Large market PlayStore

Table 3. DAFO analysis

3.4. Planning

Once the tasks that have to be done are established, there has been conducted the planning of the project within the time given. In order to achieve all the objectives previously established, I have divided my time into three different parts: Starting, development and ending an application.

Then I have defined a calendar where each task is established as well as the time that will be focused on each of them. This is the best way to acquire continuous work and no mistakes. Finally, it is important to highlight the fact that, some previous days to handling in the project, there will be a period of time to correct any problems and improve anything if necessary. Here there is a chart that summarizes the planning previously explained.

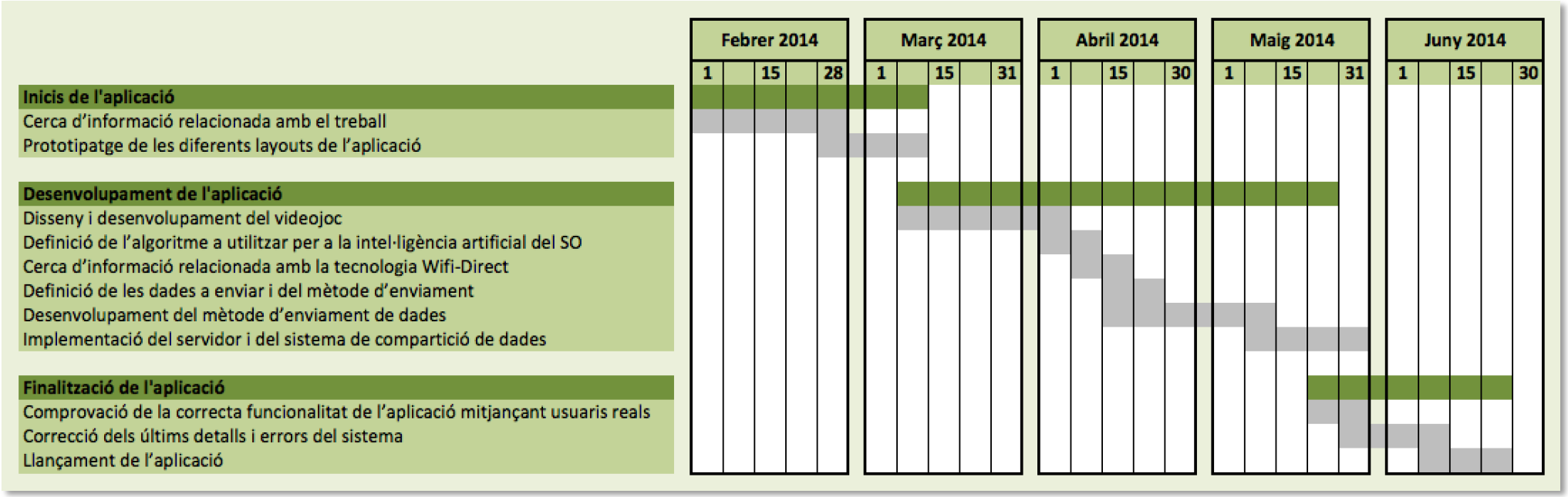

Figure 7. Project planning

3.5. Budget

The development of the application is characterized for being low cost as only the time spent by the developer should be counted. But, in order to do a more accurate budget, an estimated calculation of the time spent by the programmer and the different tools used. The following chart shows the initial cost of the project:

Description	Cost	Total
Starting the application		
Studying and prototyping the application (55 hours)	10€/hour	550€
Developing the application		
Nexus 5 mobile phone device (2 units)	349€/unit	698€
Laptop	750€	750€
Designing and developing (127 hours)	10€/hour	1270€
Ending the application		
Studying its functionality and launching (58 hours)	10€/hour	580€
		3848€

Table 4. Project budget

The calculation of the expenses budgeted based on the application will be held in the University of Lleida, so eliminates the cost of rent, electricity and water. In addition, the software used is free and in particular for the case we are also free. This is very important because it removes all kinds of budget programming tools because it costs nothing even remember that free software is not freeware equivalent. Moreover the initial budget will be maintained throughout the project, in case of need to change it will be reviewed again the viability of the project.

4. Used technology and state of the art

4.1. Platform used

4.1.1. Platform *Android*

Android is an operative system initially thought only for mobile phones, like IOS, Symbian and Blackberry. What makes it different to the others is that it is based on Linux, a free operative system core, multiplatform and free. This system lets the user program any application into a Java variation called *Dalvik*, which will be explained in section 4.2.2. The operative system gives all the needed interfaces to develop applications that access the telephone functions (GPS, phone calls…) in a very easy way using a well-known programming language. I am talking about Java.

Android used to be a practically unknown mobile phone operative system since 2005, when Google bought it. Since November 2007 it was all only rumors but, then, the Open Handset Alliance was launched. It grouped a lot of mobile phone business, chipsets. Google provided the first Android version as well as the SDK so that programmers could start their application for this system. The beginnings were a bit slow because the operative system was launched before the first mobile phone using it. But from that moment on, it quickly evolved and now is the first operative system in the world ranking. This position was achieved during the last term of 2010 [7].

One of the best characteristics of this operative system is that it is completely free. This means that the user does not have to pay anything to use it or include it in a mobile phone. This makes it be popular between manufacturers and developers as it makes the costs of launching the mobile phone or application very low. Anyone can download the font code, inspection it, compile it or even change it. This gives security to the users, as errors in open codes are quickly detected.

4.1.1.1. Architecture

The Android system is composed by five top modules: Linux Kernel, libraries, Android Runtime, Application Framework and applications.

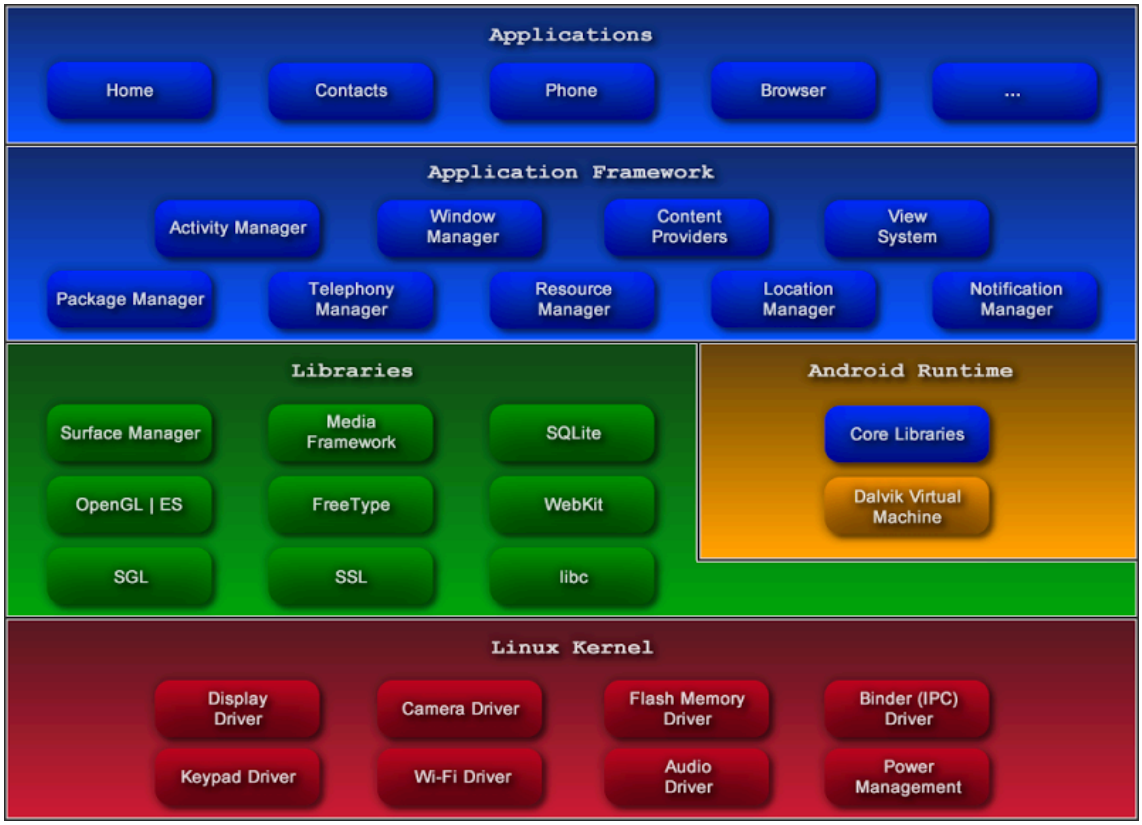

Figure 8. Android architecture [22]

After observing the structure graphic, I will specify on the Android components below:

- **Linux Kernel:** The Android operative system core provides an abstraction layer that allows the phone components, characteristics or model to be accessed. Other functions that Kernel does are: phone resources management (power, memory…) and operative system management (processes, communication elements…).

- **Libraries:** The layer located above Kernel is composed by the Android native libraries. These libraries are written in C or C++. Their function is to provide functionality to the applications and guarantee the highest efficiency in those tasks repeated at a certain frequency.

- **Android Runtime:** In fact, it cannot be considered as a layer itself because it is also composed by libraries. Its main component is the virtual machine Dalvik, which executes the non-native Android applications. These are compiled in a specific format for the virtual machine Dalvik. This ensures the fact that the application will be able to be executed in any Android dispositive that meets the version requirements. During the compiling process, the usual .class files are generated but these files are directly converted into .dex files (Dalvik executable). It is an optimization measure as .dex files take 50% less space, which saves space and accelerates the downloading processes [8].

- **Application Framework:** It is composed by the types and services that the applications use to work. The majority of their components are Java libraries that access the resources through the virtual machine Dalvik.

- **Applications:** It is composed by the applications of the device. All of them use the same application framework to access the services given by the operative system. This means that there can be invented some applications that use the same resources than the native ones. This is one of the best Android virtues: total control by the mobile phones software user.

4.1.1.2. Structure of an application

Now that we know the architecture of the operative system, it is turn to learn about the structure of an application. Below, there will be an explanation about its main components:

- **Services:** They are a type of components executed at a second level. A service is an application that runs automatically without interacting with the user. They develop important tasks for the other applications or system. Some examples of services would be: web servers, communication systems, antivirus… Services are not independent in the Android system. They run at the same process than the application that consumes them.

- **Activities:** They code functional units that do a specific task: they show a user interface to facilitate the interaction. The activities can be found at full screen, inside another application, as suspended windows… Activities have a cycle of life.

- **Broadcast receivers:** As the services do, these do neither present an IU (Interface User). They are used for an application to be able to answer to a received event like new data or a change in the net status.

- **Content providers:** It is a mechanism provided by Android that allows to store data in the device. This facilitates the data sharing between the different applications.

For the correct execution of all of these items, a control mechanism is needed. This mechanism is the AndroidManifest.xml. file. This file is needed in all the applications. It is located in the application root directory and acts as an application implementation descriptor.
Other very important items in the applications are: Intents, which are used for sending text messages or communication items between components, applications or the very same application (activities). Views, which are used to design the interface in charge of showing, through an activity, what is previously created. Notifications, which are messages that are shown to the user in order to inform or warn him of something and more.

Internally, each screen of the user interface is represented by a type of activity. Each activity has its own cycle of life. An application are one or more activities plus a Linux process that contains them. In Android, the cycle of life of an activity is not attached to only one process.

Processes are activity containers but for one use only. This means that once the activity is finishes, the process does too. During their life, each activity of an Android program can be found in any of the states that are shown in the following picture:

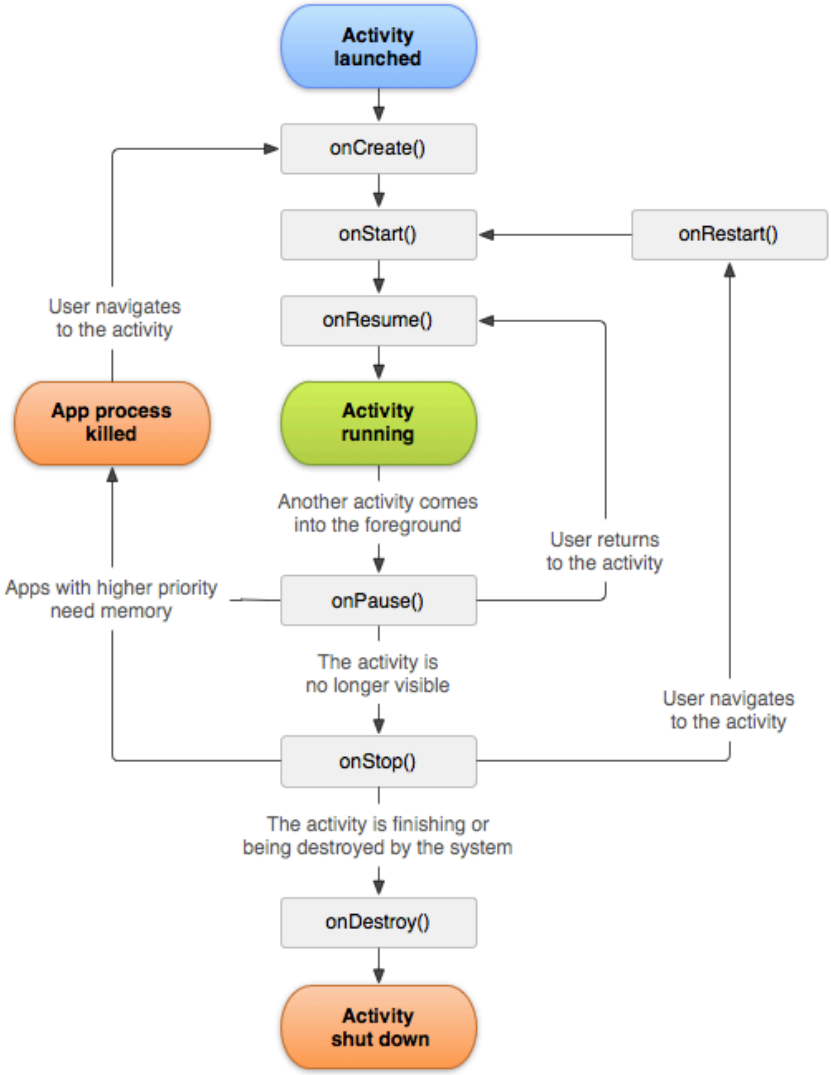

Figure 9. Androd lifecycle [23]

The developer has no control over the state in which its program is found. But it receives notifications when the state changes through these methods [9]:

- **onCreate():** This method is used when the activity is started for the first time. It is used for doing individual initializations as creating the user interface.
- **onStart():** It is needed after starting the application or after resuming the activity. If they are needed, it means that the activity is going to be shown to the user.
- **onResume():** This is needed when the activity is about to be shown to the user.

- **onPause():** It is executed when the activity is about to get to a second level, normally because another activity has been executed at first level. Here it is necessary to save the persistent state of the program.
- **onStop():** It is used when the activity is no longer visible by the user and will not be needed for a period of time.
- **onRestart():** If this mode is executed, it means that the activity is being shown again to the user from a stopped state.
- **onDestroy():** This is used right before the destruction of the activity.
- **onSaveInstanceState ():** Android will need this method for letting the activity save the instance state.

These methods are automatically used by the system and execute the internal code developed by the programmer. This is why they are called callback methods.

4.1.1.3. Versions

After having understood the structure and Android's way of working, it is time to study the main Android versions in order to choose the best one for the project. The main versions or most popular ones are the following [10]:

Figure 10. Android versions [24]

- Alpha (Android 1.0 API level 1):

  It is the first Android version. It was never commercially used, so this platform will not be developed in this project.

- Beta (Android 1.1 API level 2):

  They did not add new functions to it, they simply corrected some of the mistakes and errors of the previous version. This is the appropriate version to choose if the motto is to develop an application compatible in all the Android devices. The disadvantage is that not a lot of users use this version.

- Cupcake (Android 1.5 API level 3):

  It is the first functional version with some users. What is new is the possibility of incorporating a screen keyboard with text predictions. The devices no longer need to have a physic keyboard. It incorporates stereo Bluetooth support so it lets the user

automatically connect to Bluetooth headphones. The windows transitions are done through animations.

- Donut (Android 1.6 API level 4):

  It allows advanced search capability in all the dispositive as well as text to speech synthesis. It supports WVGA screen resolution. It appears a new XML attribute, onClick(), that can be specified on a view. Android Market is improved letting the user have an easier applications' search.

- Éclair (Android 2.0 API level 5):

  It has a new function that lets the user synchronize adaptors in order to connect it to any device. It offers a central handling purchases service. Finally, it has a brand new navigator interface and supports HTML5.

- Froyo (Android 2.2 API level 8):

  The improvement in the application execution speed is the most outstanding characteristic. This is achieved with the introduction of a new Dalvik JIT compiler. The applications development involves the following news: There is the possibility to ask the user if he/she wants to install the application in an extern storage media (as an SD card). Connections are also an improvement: from this point on it is possible to give internet connection not only to the device itself but to other devices using USB or Wi-Fi.

- Gingerbread (Android 2.3 API level 9):

  It has a new user interface with an upgraded design (a multitactile screen keyboard). It includes native support for multiple cameras, thought about the second camera used in a videoconference. The virtual machine Dalvik introduces a new garbage collector that minimizes the pauses of the application. This helps to ensure a better animation and the increase of the answer capacity in games or similar applications.

- Honeycomb (Android 3.0 API level 11):

  In order to improve to improve the Android experience in the new tablets, the 3.0 version is launched. It is optimized for devices with bigger screens. The new user interface has been completely redesigned with new paradigms for interaction and navigation. Fragments outstand among the introduced news. They can be used to

design different user interface items. 2D/3d graphics are improved thanks to the OpenGL renderer machinery accelerated. Dalvik has been optimized for allowing multiprocess which allows a faster application execution even with the single thread ones.

- Ice Cream Sandwich (Android 4.0 API level 14):
  The main characteristic is the unification of two previous versions (2.x for phones and 3.x for tablets) in only one compatible with both mobile phones and tablets. There is a new Internet data traffic manager. The NFC communication API is improved and there appears the integration with social networks.

- Jelly Bean (Android 4.1 API level 16):
  Several improvements are introduced to Google Search. Search with voice is enhanced with profile shaped results. The Google Now function allows the use of position, address book and time information in searches. There is a new international users' support as a bidirectional text and keyboards. In order to improve security, all applications are encrypted. Partial application upgrades are also allowed.

- KitKat (Android 4.4 API level 19):
  One of the main objectives of version 4.4 is to make Android available for a wider amount of devices, including those with only a 512MB capacity of RAM memory. There are also more visible improvements as the new characteristics of the user interface. WebViews (user interface components to show webpages) are based in Google Chrome software and can now show contents in HTML5. Sensors are also improved in order to minimize their consumption and a step counter sensor is incorporated.

4.2. *Wifi Direct*

Wi-Fi-Direct is a certification program that allows connecting two wireless devices without the need of an access point. This facilitates delivery of information between two devices. Normally, everybody has a router at home. Through this router, all the devices (smartphone, Smart TVs, consoles…) are connected to the Internet and to each other (if possible).

4.2.1. Technology description

If not a lot of devices are connected to the router it works pretty well, but when there are a lot of them it can cause problems. At the same time, not everybody have routers so it is more interesting to be able to directly connect to the Internet without the use of a router.

This is where Wi-Fi-Direct provides value. Compatible computers are able to create an *ad-hoc* network (point to point), creating an access point via software. A new function that is being generally integrated in televisions allows to share the content of our device into it. What is more, the compatibility of both devices with Wi-Fi-Direct is not needed if, at least, one of them is.

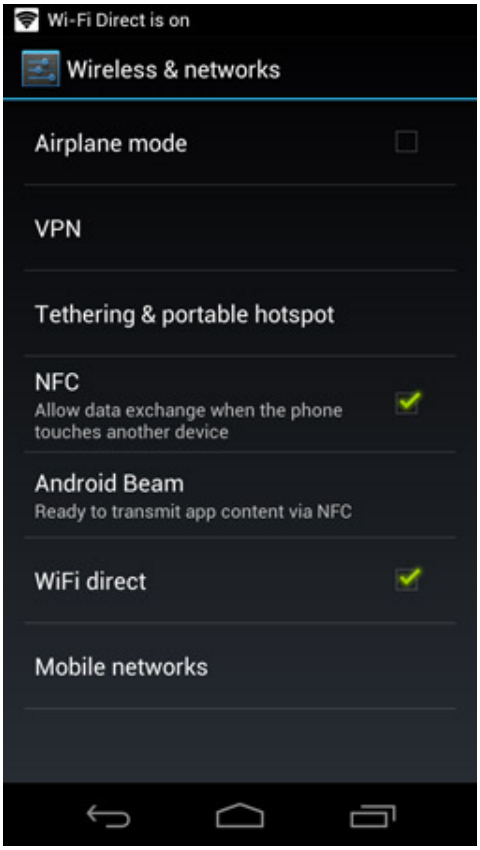

Figure 11. Wifi Direct on Android

Android mobile phones from 4.0 version above have Wi-Fi-Direct serially included. Apple devices, thanks to IOS, have their own Wi-Fi-Direct called airdrop from version 7.0 which is only compatible with other IOS. Wi-Fi-Direct uses some rules to accomplish its functions:

- **Wi-Fi:** Wi-Fi-Direct uses the same Wi-Fi technology than enabled devices use to communicate with wireless routers. A Wi-Fi-Direct dispositive can work essentially as an access point so those devices habilitated for Wi-Fi-Direct can directly connect to it. This was previously possible thanks to the creation of ad-hoc networks but Wi-Fi-Direct improves this characteristic with an easy configuration and discovery characteristics.

- **Devices' Wi-Fi and discovery services:** This protocol offers connection using the devices' Wi-Fi in order to connect with each other and check the services supported before connecting. For instance, a Wi-Fi-Direct device could see all the compatible devices supported in a zone and then reduce the list only into the devices that allow printing (like Wi-Fi printers).
- **Wi-Fi Protected Setup:** When two devices connect to each other, they automatically connect through Wi-Fi Protected Setup or WPS [11]. We only have to wait for devices' manufacturers to use a secure connection for this WPS connection, as it is the device what has to support this type of protection.
- **WPA2:** The devices use WPA2 encryption, which is the most secure way of encrypting Wi-Fi.

4.2.2. Key features

WiFi Direct technology differs from other connection protocols for a number of unique features. The main thing is that WiFi Direct devices can connect to each other without going through an access point, ie you do not need to use a router. This is due to the fact that WiFi Direct devices establish their own ad-hoc networks, when necessary, allowing you to see what devices are available and choose how to connect.

In order that the operation of devices with WiFi Direct always have the same characteristics and standards, there is the association responsible for WiFi Alliance certified wireless devices. This certification means that any device with WiFi Direct technology is guaranteed to work with any device that accepts the same technology.

The idea behind is that WiFi Direct connections require simple tasks simple. One example is the new printers capable of printing both from a computer to a phone. In addition, there are devices capable of sharing images between different rooms in a house, or even send videos directly on TV.

It should be called that in the same way that protects the home network password to prevent potential intruders, WiFi Direct also has its own security. To avoid this, use WiFi Direct WiFi Protected Setup WPA2 and to prevent unauthorized connections and maintaining their private communications. To also achieve security between devices, often linking devices can be done

in several ways; physical buttons "press Gadget X and Y then the same gadget" with PIN codes, QR codes and even NFC.

WiFi Direct includes a potentially useful facility; automatic service discovery. All devices not only know the existence of available devices but can also detect what devices are nearby and offered each of them. This means that if you try to display an image, only see devices that can display such images. If you want to print, only the devices that will be connected printers.

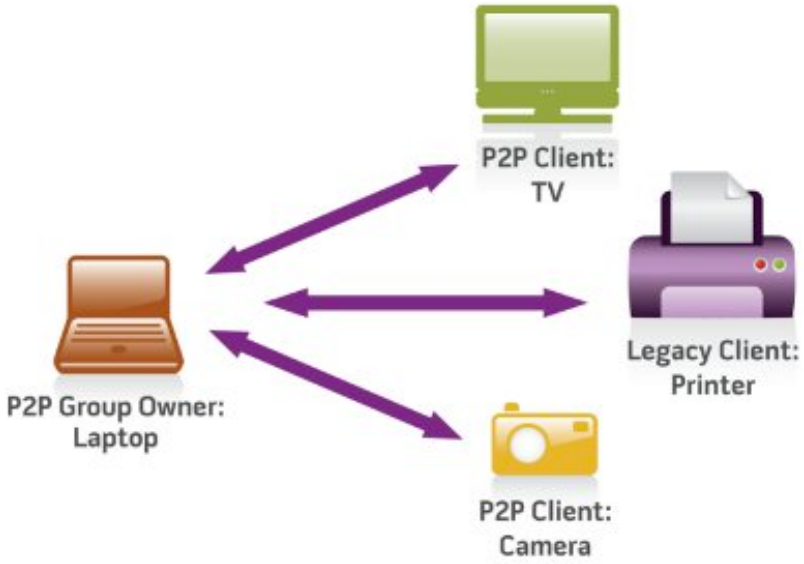

Figure 12. Connection Wifi Direct devices [25]

Another feature is that the manufacturers do not need to add additional radio transmitters and receivers to operate: the idea is to have WiFi Direct, as part of the standard wireless radio. It is completely compatible, so there is no problem in connecting devices together. The alliance focused on Wifi now claims more than 1,100 devices have been certified since October 2010, including TVs, smartphones, printers, PCs and digital tablets. [12]

WiFi Direct WiFi can also be referred to as peer-to-peer or P2P Wifi, operates as an equal. In WiFi Direct devices connect directly to each other instead of using a wireless router. While WiFi Direct is assumed theoretically that should be a standard for connecting multiple devices, today there is still no 100% functional application that allows such connections [37].

For example, you have two laptops, each announced support for WiFi Direct. It is possible to assume that there would be a way to set up file sharing between computers with ease using WiFi Direct, but for now it is not so (except Apple's AirDrop technology). Nor is there an easy way to connect an Android phone to a Windows laptop and actually both use the same

technology. In the future, WiFi Direct can become a more useful standard and set aside wired networks.

4.2.3. Operation

This technology, despite being in a lot of devices as shown in chapter 4.2.4., It is still unknown to many users. In order to properly understand the functioning, it will show an example of operation. Imagine that you are at home and you've got the router working and emitting wireless signal. The router is the center of the trunk in a tree with many branches and these branches can interact with each other, but always through the trunk of the (router). Below is a graphic example [13]:

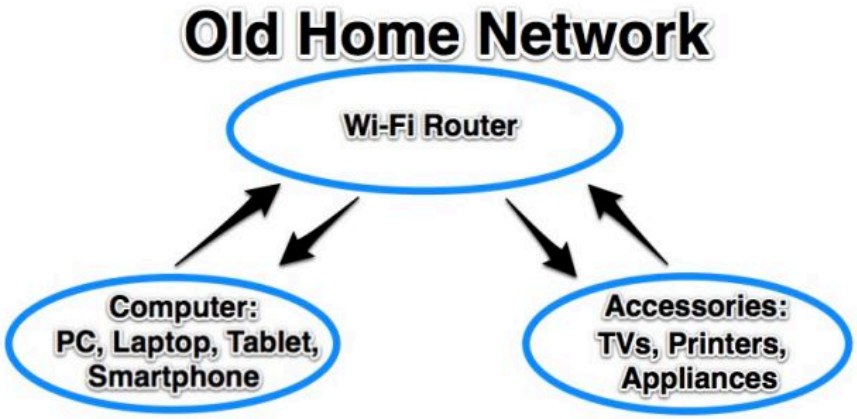

Figure 13. Classic home network [26]

In this case the router is essentially the gateway connectivity, called wireless access point (WAP). Use WiFi Direct means cutting the trunk network and connect branches between them without an access point intermediary. The configuration of the connection between these devices usually being very simple because normally you only need to press a button or enter a password. Thus, WiFi Direct Bluetooth works as simply a longer range and greater stability [38].

For example, if I connect my PC, laptop, digital tablet or smartphone to a compatible printer with WiFi Direct, I just have to click a button once the printer to identify my device (my PC for example) and connect. A computer, a dialog box will appear to confirm the new connection correctly. This connection is accomplished by creating a connection between two devices using a protocol known as WiFi Protected Setup (WPS). The standard WiFi Protected setup is what makes connecting simple devices and the ability to add devices to a traditional wireless access point. WPS was originally created so that users could easily set up their wireless networks

without being technical wireless security. Following the previous example, once cut the trunk of the tree, would graphically as follows:

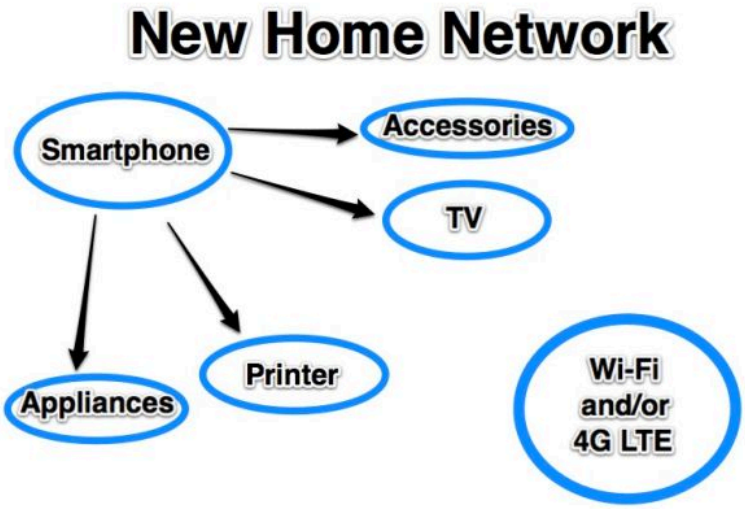

Figure 14. Network with Wifi Direct [27]

Use the WiFi Direct technology means making use of a simple technology with a variety of simple and complex functions (a wireless mouse is simple, a smartphone can share your hotspots, media files and streaming is complex). It is for this reason that with this functionality, WiFi Direct technology is very much expected in the near future in all kinds of applications and devices.

4.2.4. *Wifi Direct* devices

In November 2011, the Digital Living Network Alliance (DLNA) announced that it was included WiFi Direct to its interoperability guidelines. Since then, Google has added WiFi direct support all versions of Android, Ice Cream Sandwich from Android 4.0.

Samsung has included WiFi Direct the device from Android 2.3 Galaxy S2, although only available only to connect to other devices from the same company. To use DLNA streaming is now common in all the latest devices launched on the market, including the HTC One, LG and Sony Xperia Z1 G2 [14]. WiFi Direct is even the new iPhone 5S already included in the operating system iOS7. Not interested in losing, has updated its BlackBerry operating system to include BB10.2.1 Wifi Direct. Even the new Xbox One comes equipped with Wifi Direct connection for connecting digital tablet or smartphone to install applications as SmartGlass to work a bit faster and much easier.

You can check all devices that have this technology in the following document online: https://www.wi-fi.org/certified-products-search

4.3. Runtime environment

4.3.1. *Java Virtual Machine*

The execution of a Java program consists in interpreting the byte codes which means transforming them into a system code and executing the code while translating into the machine code. The JVM (Java Virtual Machine) is the one in charge of doing this chore. Luckily, the code is not in a very high level language. Byte codes are a written intermediate code for the JVM instructions game. What is more, the virtual machine can use JIT (Just In Time Compiler). They translate the byte code into a native code and every time that JVM needs this code, JIT provides it the pre-executed one [15].

Before JVC starts this interpretation process, it is necessary to do some chores to get everything prepared for the program to execute. At this point, the security policy is implemented. There are three different components in this process:

- **Class loader:** This item separates the classes in order to avoid any attack. In Java 2 there are three class groups associated to different route searches: system classes (associated to the boot class path), extension system path (associated to the extension class path) and user or application class (associated to the user class path). The searching order is: system, extension and user. Once a class is found, the search stops (an application could only replace a system class modifying the boot class path and this is not possible without a free system access). The class loader is a type of Java and can be expanded in order to define special classes but only for applications. If an applet could define its own loader it could modify the system loader and start the machine in which the navigator is executed [16].

- **Class file verifier:** The function of this component is to validate the byte codes. Even though this chore can seem nonsense, it is not, as not all the byte codes are always correct because they can be hand-made or using modified compilers. The system distinguishes two types of codes. The code in which it relies on (generally system classes or the ones validated by the user) and the code in which it does not rely on. The first type are not validated. The class file verifier is part of the JVM and, can only be replaced by changing the virtual machine.

- **Security manager:** It is the one in charge of checking the runtime access. It is a system class and can be expanded through the applications. The Java 2 default implementations define a system based on access policies.

In the following image you can see a summary of the whole process:

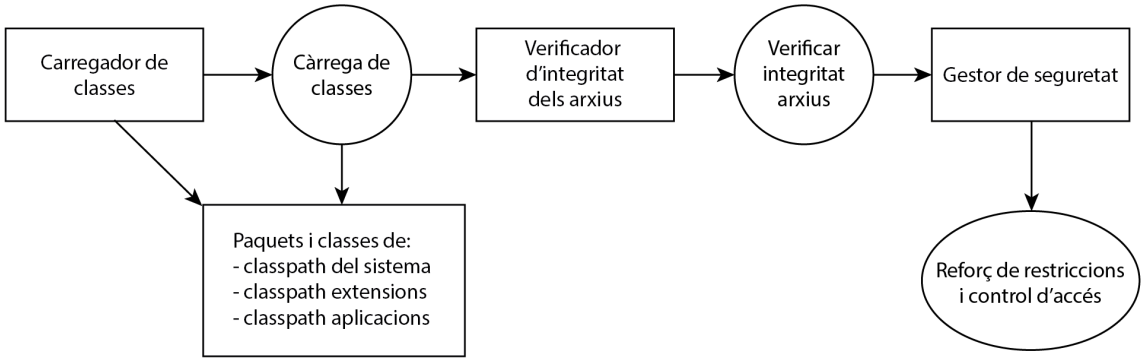

Figure 15. JVM scheme [28]

4.3.2. *Dalvik*

Even though it is not supported by the libraries previously mentioned, the Android execution environment is not considered a layer itself as it is also formed by libraries. Specifically, by the essential Android libraries which include the major functional part of the usual Java ones as well as other specified ones for Android. It is based on Kernel functionality (like threading or the low memory level management) [17].

The main Android runtime environment component is the virtual machine Dalvik which is a component that executes every non-native Android applications. Applications are usually codified in Java and then compiled but not to generate an executable binary compatible with the Android's specific architecture machine. Instead of this, they are compiled in a specific format for the virtual machine Dalvik which is the one that executes them. This allows to compile the applications and distribute them once compiled. They are guaranteed to be executed in any Android device that has the minimum version of the operative system of each application.

Dalvik must not be confused with the virtual machine Java as they are two different virtual machines. In fact, Google created this virtual machine to avoid license problems. Java is only the programming language and the generated programs are not compatible in byte code Java.

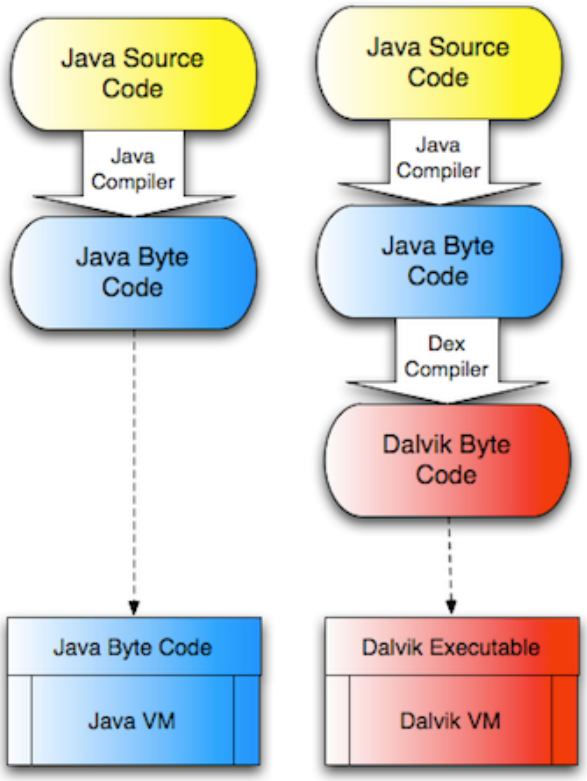

Figure 16. Dalvik running scheme [29]

The files generated by this virtual machine, .dex files, are a lot more compact because of the limitations of the devices which they are made for (up to 50%) than the generated by a traditional Java virtual machine. In order to achieve this, Dalvik is based on registers instead of a stack to store data, which requires less resources.

Android applications are executed in their own instance of the virtual machine Dalvik. This way, interferences are avoided and there is access to all the libraries mentioned before and, through these libraries, the machine and the other resources managed by Kernel.

4.4. Development tools

Once the execution environment has been studied, the three Android system main tools (Eclipse, Android Studio and IntelliJ) are studied. After all the information found and after using the three tools to do the first examples, I decided to implement the project with Eclipse.

4.4.1. *Eclipse*

Eclipse is a developing platform, designed to be expanded through plug-ins in an undefined way. It was conceived from its origins to be a development tools integration platform. It does not use a specific language as it is a generic IDE although it is very popular among the Java language developers using plug-in JDT which is included in the standard distribution of IDE. It provides tools for the work spaces management, writing, unfold, execute and debugging applications. As basic items, an IDE has a code editor, a compiler/interpreter and a debugger. Eclipse works as a Java IDE and counts with a great amount of software development tools. It also supports other programming languages, as C/C++, Cobol, Fortran, PHP or Python. Extensions (plug-ins) can be added to the Eclipse base platform in order to expand its functionality [18].

The word Eclipse identifies the free software community for the Eclipse platform development. This work is divides in projects which objective is to provide a strong, scalable and quality platform for the software development with IDE Eclipse. This work is coordinated by Eclipse Foundation. This foundation is a nonprofit foundation that gives support to both Eclipse ecosystem and community.

The main characteristics are:

- **Perspectives, editors and sights:** The Eclipse work concept is based in the perspectives, which is nothing else that a windows and editors pre-configuration and related to each other. They allow to work in a specific work environment in an optimus way.

- **Project management:** Eclipse development is based on projects, which are the group of resources related to each other as font code, documentation, configuration files, directory trees… IDE will provide assistants and support for the project creations. For instance, when one is created, the right type of project's perspective being created will open with the sights collection, editors and windows pre-configurated by default.

- **Code debugger:** It includes a powerful debugger which is easy to use and intuitive. It can help us to visually improve our code. For this to happen, we have to execute the program in debugger way (with only a button). Again, we hace the specific perspective for the code debugging, the debugging perspective, where all the needed information for this chore is orderly shown.

- **Extensive plug-in collection:** They are available at huge amount, some published by Eclipse and others by other. Because of the fact of having been factor standard for a lot of time (not the only one but one of them), the available collection is enormous. Some are free, paid, under different licenses but almost for everything there is the right plug-in.

As the use of JDT is so expanded, we honor it by giving it a specific section. It is the plug-in in charge of IDE Java language support, included in the Eclipse default standard version that, as I have explained before, it cannot support a specific language.

When we open a Java project, the corresponding view opens. It is formed by two sights: Outline and Package explorer. The Outline sight is the one in charge of showing the open class scheme in the active editor of the moment. A very interesting fact is that when we have an active sight, extra icons are displayed on the tool bar which will let us have a faster access to the most used functions of the sight.

The editor coloring code is a very interesting characteristic, doing for this the syntactic recognition of all the words reserved in Java language. In the same way, it allows to automatically complete the code (code completion) with context depending suggestions that allows to write the code faster. The code format will be able to be configured, the comments way of being written, including comments for the later Javadoc creation. We can create class skeletons automatically as well as getters and setters method generators and a lot more that, from nowadays point of view, can seem typical but useful.

Eclipse origins are found in its ancestor IBM VisualAge, which developed a dual virtual machine for Java and Smaltalk (language in which the project was written). When Java started to expand and its popularity increased, IBM decided to abandon the dual virtual machine project and develop a new platform based on this language. In 2001, together with Borland, the

Eclipse foundation was born, nonprofit and making Eclipse become an open code project licensed by Eclipse Public License. This foundation has enriched with the inclusion of important businesses of the development world: Red Hat, Oracle, HP…

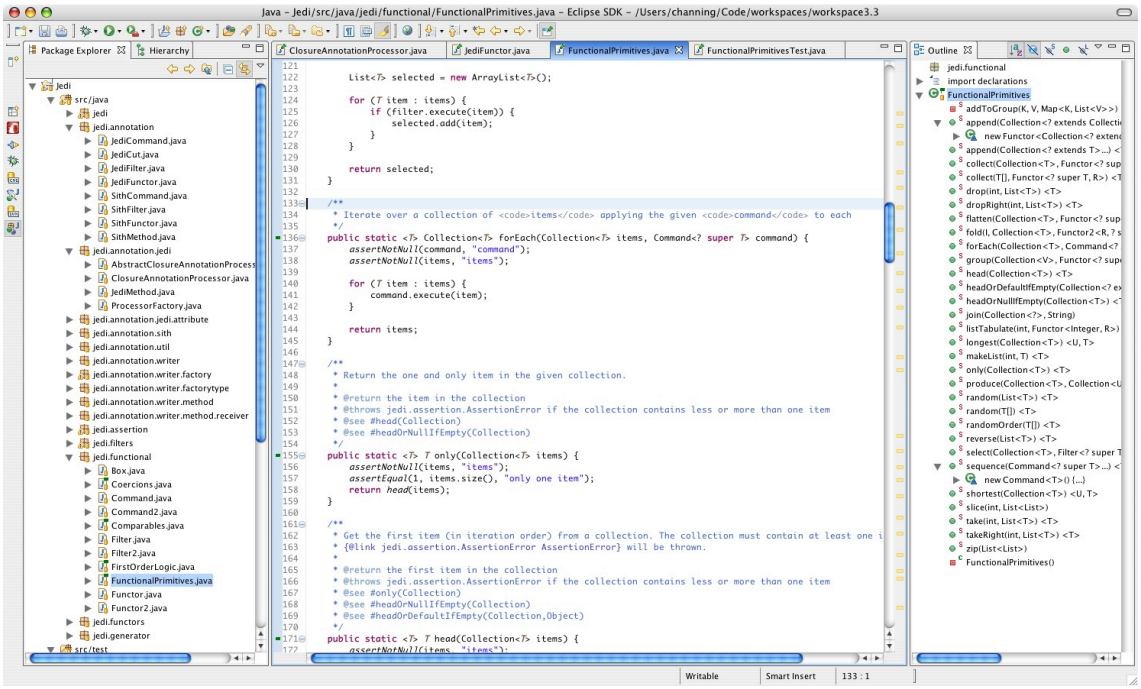

Figure 17. Screenshot Eclipse program

5. Requirements analysis

The global analysis of the requirements of an application is a process of conceptualization and formulation of the main concepts involved when developing an application. Is a fundamental part of the development process of an application, since most software defects originate in the requirements analysis phase, and they are more expensive to repair. After performing a search for information analysed the different application requirements in order to see the feasibility of the project.

5.1. Functional requirements

Functional requirements are eservices declarations that will provide the system the way in which this will react at particular entrances. In some cases, the functional requirements of the systems do also declare what the system must not do.

Many of the software enginery problems come from the imprecision of the requirements specification. For a system developer it is natural to give interpretations of an ambiguous requirement with the finality of simplify its implementation. However, this is not what the customer wants. New requirements must be stipulated and system changes must be done, delaying its delivery and increasing the cost.

The functional requirements specification of the work in process are the following:

R1. Basic requirements

- R1.1. Show the possibilities of play
- R1.2. Allow play all types
- R1.3. Record player scores
- R1.4. Show data sets completed
- R1.5. Having an artificial intelligence module

R2. Requirements connection

- R2.1. Get connection between two devices
- R2.2. Identify users of the game
- R2.3. Get information about the player (user)
- R2.4. Get statistics on completed games (*ratio*)

R2.5. Show information needed device adversary

R2.6. Get data using the server

R2.7. Provide networking

5.2. Non-functional requirements

They are those which are not directly referred to the specific functions that the system delivers but to the emergent propriety of it as reliability, time answering end storing capacity. Alternatively, they define the system restrictions as the input/output devices capacity and the data representation used in the system's interface.

The non-functional requirements come from the need of the user, because of the budget restrictions, political organizations, the interoperability need to other system software or external factors as safety regulations, privacy policy and more. The non-functional requirements specifications of the work in process are the following:

R3. Requirements development

R3.1. Certificated for Wifi Direct directives

R3.2. Show a smooth and fluid game

R4. Design requirements

R4.1. To facilitate use of application interfaces with simple buttons clear and easy control of the game

R4.2. Develop the game using the original design

R5. Security requirements

R5.1. Encrypt data when exchanging information with other devices and server using password

R5.2. Avoid crashes server

R6. Functional requirements

R6.1. Store user information in the last 20 plays

5.3. Use case diagram

The use case model describes the functionality proposed by the new system. A use case represents an interaction unit between a user (human or machine) and the system. A use case is significant simple working unit like, for instance, “system validation”, “system register” and “order creation”. In this project, the use case will be distinguished in three different diagrams; playing against the device, playing using Wi-Fi-Direct and playing using a server.

5.3.1. Playing against the device diagram

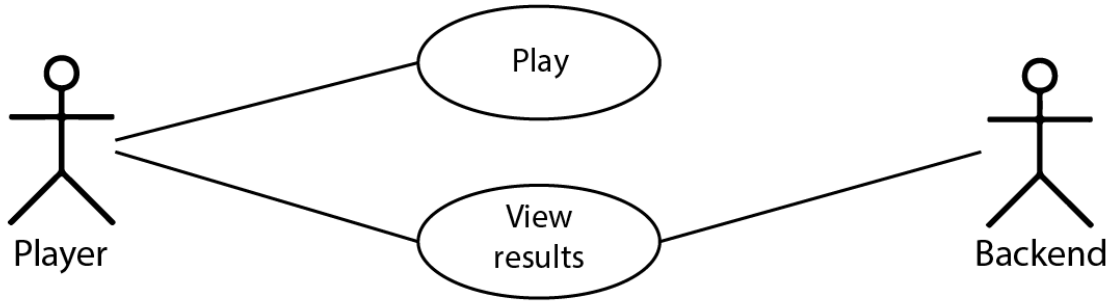

Figure 18. Use case diagram: Play against the device

5.3.2. Playing using *Wifi Direct* diagram

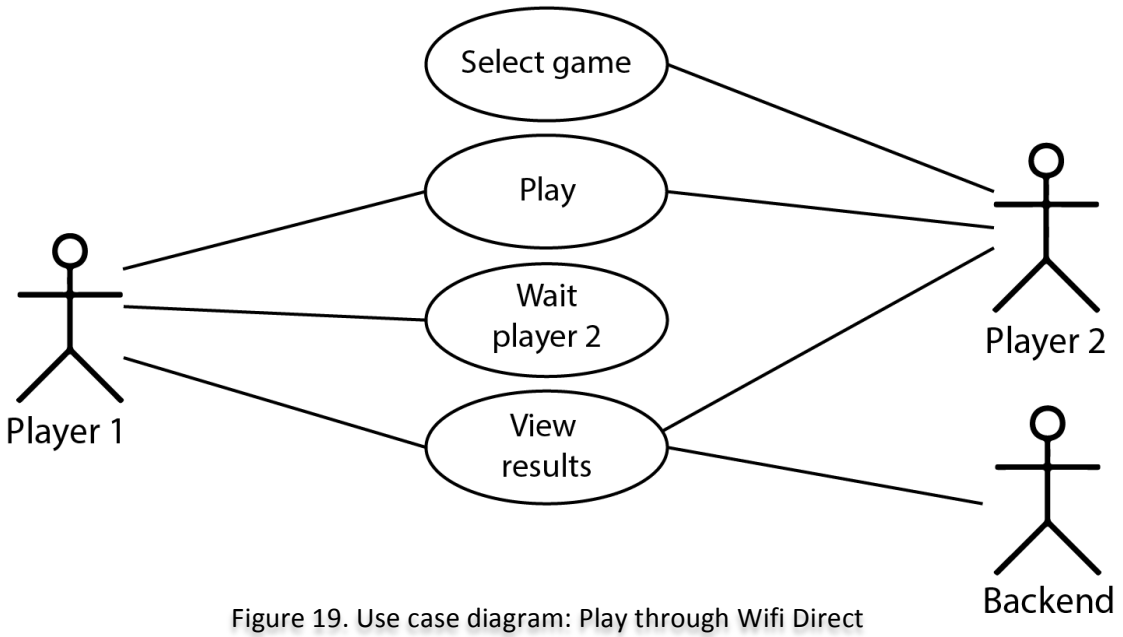

Figure 19. Use case diagram: Play through Wifi Direct

5.3.3. Playing using a server diagram

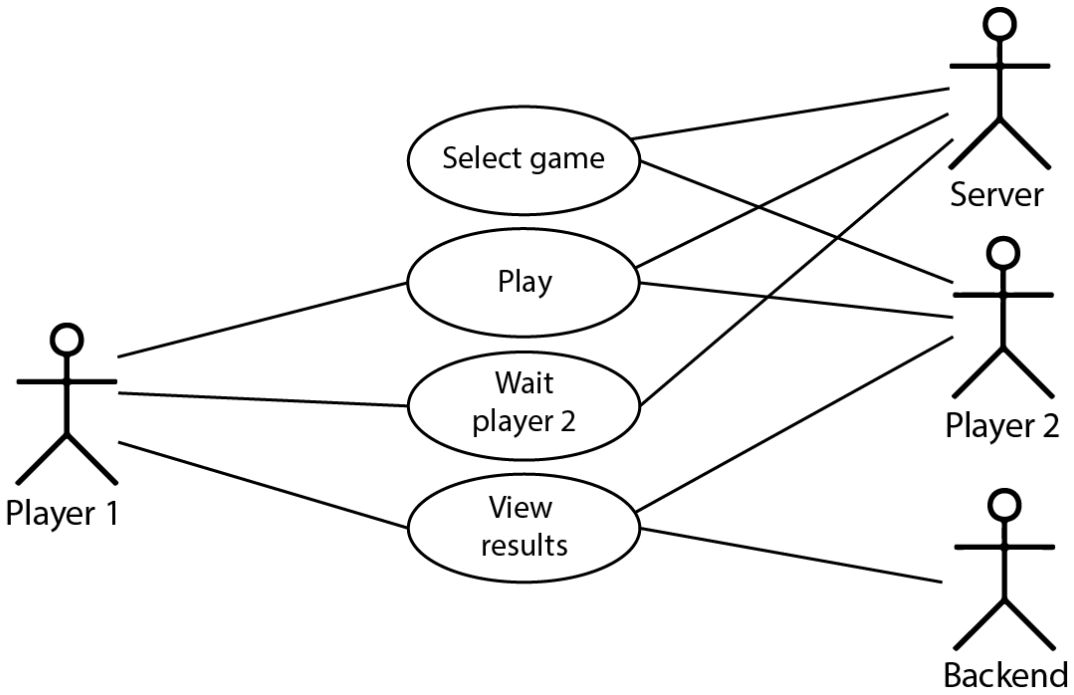

Figure 20. Use case diagram: Play through a server

5.4. Specification of use cases

Then specify the different use case diagrams shown in the previous method based on the game:

High-level specification

Use case: Play (*Wifi Direct*)

Actors: Main: Player 1

Secondary: Player 2

Purpose: Start a new game

Description: A user creates a new game and start it. At this time the user expects the second player to connect to the current game. The game begins when both players are the same game. The game ends when one player loses all three balls throughout the game.

Specification expanded

Cross references:

Requirements: R1.2., R1.3., R1.5., R2.1., R2.2., R2.3., R2.5., R3.1., R4.2.

Typical course of events:

Action of actors	System responses
1. The user selects the type of game.	
	2. The system generates a new set of guidelines based on the user.
3. The user expects the connection of a second player.	
	4. The system keeps waiting for a user to connect to the game.
	5. The system received a request from a user inserts the game and this begins.
6. The user observes that there is a second player and the game begins	
	7. The system expects to deplete the life of the game and sends a message to the winner of the game.
8. It displays the username winner.	

Alternative courses:

3.a. It connects any user or connection failure.

1. The system sends a warning message and close the access point.

High-level specification

Use case: View results

Actors: Main: Player 1

Secondary: Player 2

Purpose: Check the results of the latest games

Description: Users query the different results of the latest games, watching the scores and games won or lost.

Specification expanded

Cross references:

Requirements: R1.3., R1.4., R2.2., R2.3., R2.4., R2.6., R2.7., R6.1., R4.1., R5.1.

Typical course of events:

Action of actors	System responses
1. The user selects check the results of past games.	2. The system generates a query to the backend of the latest games.
	3. The system will list the results on screen.
4. The user selects a room play.	5. The system sends information about the selected item.
6. The user returns to the home screen.	

High-level specification

Use case: Wait for player 2 (*Wifi Direct*)

Actors: Main: Player 1

Secondary: Player 2

Porpuse: Wait for the connection of a second player to start the game

Description: Once created the game, the user is waiting for a user to connect opponent to start the game.

Specification expanded

Cross references:

Requirements: R1.2., R2.1., R2.2., R2.3., R3.1., R3.2., R4.1.

Typical course of events:

Action of actors	System responses
1. Player 1 creates a new game.	
2. It is waiting for player 2.	
	3. The system sends the new game created.
	4. The system expects to request player 2.
	5. The system confirms the second player and the game starts.
6. The player starts playing.	

High-level specification

Use case: Select game (*Wifi Direct*)

Actors: Main: Player 2

Secondary: Player 1

Purpose: Select an existing game

Description: Player 2 selects in a list, the name of the game in which you want to join and play against one player.

Specification expanded

Cross references:

Requirements: R1.1., R2.1., R2.2., R2.3., R3.1., R3.2., R4.1.

Typical course of events:

Action of actors	System responses
1. Player 2 items available query.	
	2. The system searches for the various items available.
	3. The system will list the possible items.
4. Chooses which game to enter.	
	5. The system sends confirmation from player 2 to player 1.
6. Start the game.	

5.5. System specifications

5.5.1. Hardware resources

Among the Android users, it is frequent to hear about disconformity because it is a slow operative system, inefficient or it consumes too many resources. The reality is that if we use a low or average range computer, the Android experience differs a lot from the users who use high range computers because Android is anything but machinery friendly. The problem with Android is not in the operative system although part of the fault is found in the users' erratic experience. But the major part of the problem is found in the machinery, as explained below.

The fact that Android is an operative system which runs a lot of chores at the same time, something that is similar to what nowadays computers do. But this very same functionality is the one that cause an excessive consumption of the system resources as like a computer, the more activity executed at the same time, the slower the system will run. IOS, Android or Blackberry 10 are different in the chore evolution. In IOS the management is done so that the attention is focused on the application being executed at a certain moment while the others are not being used (even if they are open) because they get to a waiting state that generates a less resources consumption.

Getting back to the android case, let's think on a supermarket line. The operation speed will of course depend on the shop assistant speed in charging the costumers but more important, in the amount of people in the line. Which is the alternative? Opening more cash registers which would improve the management but at the same time, it would need more workers. In the case of an operative system this would be in more consumed resources. This is the cost of applying multitask in Android. Android is two steps better that IOS or Windows Phone, whilst Blackberry 10 is found in the middle. If the device has enough RAM memory and an adequate processor, the operative system will run without any mistake.

The good news is that Android 4.4 is optimized for being used in devices of even 512MB RAM and other less favored specifications. Google is planning to change the virtual machine Dalvik for the new virtual machine ART in the future which will avoid the existent gap between low and high range. For this reason, the running project will be applied over operative systems with a more upgraded SO as the cost of multitask is essential when sending or receiving data with Wi-Fi-Direct or a server.

5.5.2. Technic restrictions

In order to use the application developed in this project, it will be necessary to take into account some mobile phone device technic restrictions:

- Minimum of 400Hz processor
- Minimum of 640x480 screen resolution
- More than 128MB of ROM or 64MB of RAM
- Android 4.1 operative system or superior
- Wi-Fi
- Networking (using Wi-Fi or 3G)

If these technic restrictions are accomplished, the application should correctly work without any errors and data fluency. In the case of the needed computer for the application development, these are the technic restrictions:

- Minimum of 1.8GHz processor
- More than 256MB of RAM
- 350MB hard drive
- Networking

If these restrictions are accomplished in both the device and the computer, the project should be overtaken perfectly.

6. Interface design

Once the objectives of the application have been defined, the design of it must be chosen. Initially I thought on maintaining the original design of Pong Original but I finally decided to give it a more upgraded image. For this reason, through some first handmade designs, a little sketch has been designed. In the next sections there is the result of the entire job done and the final design for the project application.

6.1. Screens and menus design

When having to design screens and menus for Android, the first problem that we find is the enormous difference that exists between one and another device not only in their performance but in resolution, which is the main problem we will find when designing.

The screen timeline supported by Android devices starts by T-Mobile G1, the first available Android device in the market and has a 320x480 HVGA screen. HVGA means “half of the video graphics array” (or VGA half size) which is the standard screen size even though nowadays it exist devices with a lot higher resolution that will be shown later on. In order to simplify, Android classifies screens (by measuring the length of a diagonal from the upper left corner to the lower right corner of the screen device) in four general sizes: small, average, large and extra-large.

320x533 pixel resolution is recommended to be used as a reference and then to introduce designs personalized for other screen sizes as it is considered as average size in Android. When talking about extra-large screen size it refers to tablet screens. However, nowadays most popular smartphones have HD screens up to 5 inches and 1920x1080 resolution (445ppp). For this reason, what was considered as average, it is not functional anymore. Although Android’s resolution is thought to be for 320x533 pixel, from now on there will be shown the main characteristics for bigger screens as this is what can be found nowadays in the market [19].

When designing in Android it is important to highlight that it is not talked about screen resolution but screen density. When talking about resolution we talk about the amount of pixel found in a screen area. Below there is the explanation how Android developers define resolution and density screen:

- **Resolution:** The total amount of physic pixel in a screen.
- **Screen density:** The amount of pixel in a physic area of the screen, normally known as DPI (dots per inch)
- **Independent density pixel (DP):** This is the virtual pixel unit that is used in a user interface design definition with the purpose of showing the sizes of the design or the density position in an independently way. The independent density pixel are equivalent to a physic pixel in a 160 DPI screen, which is the reference density assumed by the system as the screen average density. In the time execution, the system controls in a transparent way any enlargement of DP units, over the base of the real density of the screen being used. The conversion of DP units to screen pixel is simple: Pixel =DP (DPI/160). For instance, in a 240 DPI screen, a DP is equal to 1,5 physic pixel. It always uses DP ones as a unit to define the designs of its user interface to ensure it will correctly be shown in screens with different densities.

Android divides screen densities in four basic densities: LDPI (low), MDPI (medium), HDPI (high) and XHDPI (extra high). This is important as it will be necessary to have bitmaps depending on the different densities. Minimally, there should exist MDPI and HDPI format images in an application in order to ensure that, in any case, images will be shown at the appropriate quality. This means that all bitmaps graphics need to be amplified or reduced based on their size (320x533). However, it also exists a way to parsing SVG files that provides a way to escalate the vector to different screen sizes and densities without losing the image quality. But in this project this functionality will not be shown.

	Low density (120), *ldpi*	**Medium density (160), *mdpi***	**High density (240), *hdpi***	**Extra high density (320), *xhdpi***
Small screen	**QVGA (240x320)**		480x640	
Normal screen	**WQVGA400 (240x 400)** **WQVGA (240x432)**	**HVGA (320x480)**	**WVGA (480x800)** **WVGA854 (480x854)** 600x1024	640x960
Large screen	**WVGA800 (480x800)** **WVGA854 (400x854)**	**WVGA800 (480x800)** **WVGA854 (480x854)** 600x1024		
Extra large screen	1024x600	**WXGA (1280x800)** 1024x768 1280x768	1536x1152 1920x1152 1920x1200	2048x1536 2560x1536 2560x1600

Table 5. Screens densities on Android [30]

The bitmaps requirement is similar to the one to prepare graphics when printing. If there is a 72 PPI image it will be too pixelated and blurred once being amplified or reduced when printing, for instance, a poster. Instead of amplifying or reducing the image, it is necessary to remake the image in vector format or use a high-resolution image and adjust the image resolution around 300 PPI in order to print it and not losing the image quality. The Android screen density works in a very similar way, with the exception that we do not change the file resolution but only its size. A good standard to adopt is 72 PPI as the weight is low and the resolution fine for medium size.

Maintaining density independency is important because, without it, a user interface item (like a button) would appear physically bigger in a low-density screen and smaller in a high-density one. Density changes related to size can cause problems in the application in design and usage.

Let's suppose now that an icon wants to be shown in 100x100 pixel bitmap in one of the main screens (remember that the base is a 320x480 layout). Placing this very same 100x100 icon in a IDPI screen device would make its appearance big and blurred. The same way, placing it in another device with a HDPI screen would make it be too small or even unappreciative (thanks to the fact of the device having more dots per inch than the MDPI screen). In the following image this can be observed depending on the type of screen:

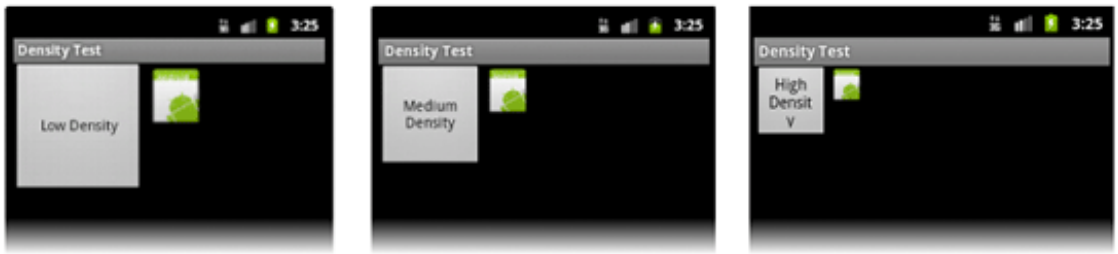

Figure 21. Bitmap example [31]

In order to adjust the images into the different devices screen densities, it is necessary to follow a 3:4:6:8 scale relation between the four density sizes. Using this and a little simple Math we can create four different versions to our bitmaps to be able to work in the application:

- 75x75 for low-density screens (x0.75)
- 100x100 for medium-density screens (base screen)
- 150x150 for high-density screens (x1.5)

- 200x200 for extra-high-density screens (x2.0)

Figure 22. Various bitmaps [32]

The same way it is important to differentiate an image depending on the screen density, it is possible to classify it in the project depending on the screen resolution in which the application is wanted to be used. A solution for being able to classify the bitmap type is the following:

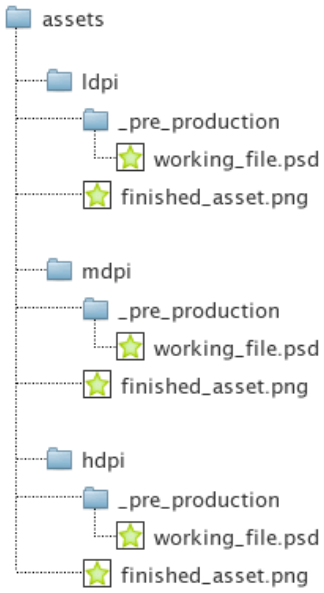

Figure 23. Bitmaps classification [33]

In the previous image it is shown the organization and called the file and directories structure. This way we can use the same name for each of the density screen files. In order not to do more work than the one needed, it will be necessary to create the images under a 72 PPI standard and escalate the images depending on the version or format in which the image wants to be shown.

Telephone and computer companies also present a great variety of Android tablets with different screen sizes. However, after having revised the most popular ones, we can conclude that both main screen sizes are focused on 1280x800 and 800x480 physic pixel.

With the Android 3.0 and on launching, Google provided the manufacturers a graphic interface specially designed for tablets, leaving apart the Back button which has been substituted by a navigation button and a system state bar located in the lower side of the screen. Android 3.0 has a fully renovated visual aspect, and incorporates all the design patterns that version 2.0 introduced. One of the huge differences in Honeycomb is a mechanism called fragments. A fragment is a component in a layout that can change its size and position depending on the orientation and screen size. This also addresses the problem of designing for different type of screens, giving the designer and developers a way of doing their components more flexible depending on the screen limitations found by the application. This way, the screen components can stretch, stack, expand, collapse, show and hide. In the following image there are examples of how to use fragments in tablets:

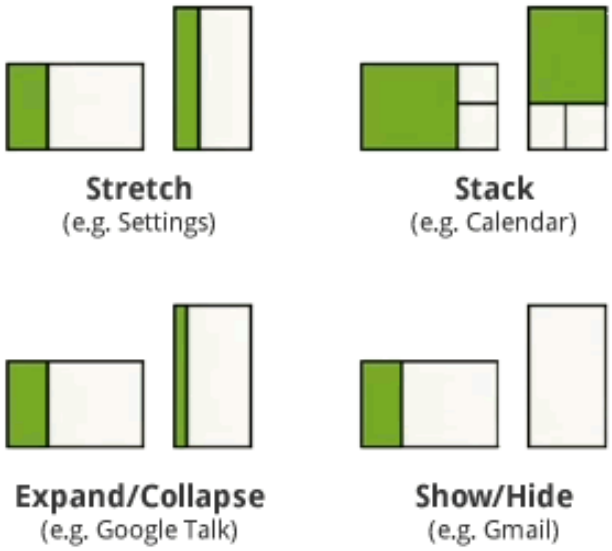

Figure 24. Fragments [34]

Thanks to fragments and to the different characteristics describes above, it will be possible to create the application right menus and screens design. The project will mainly be developed for smartphones but thanks to the information obtained from the screen design it will be done in order to use the same application in bigger devices like tablets.

6.2. Menus design

When designing the different menus for the application, it has been first analyzed the operation and characteristics of the possible menus. For this reason, a very interesting resource called Android Patterns has been used. It has been born from the analysis of different Android applications and there is very interesting information about the designing patterns there. This resource is made for interaction designers that are getting started with Android and need to have doubts solved about design and behavior patterns given in the applications. Here there are the different patterns for Android characteristics and classifications [20]:

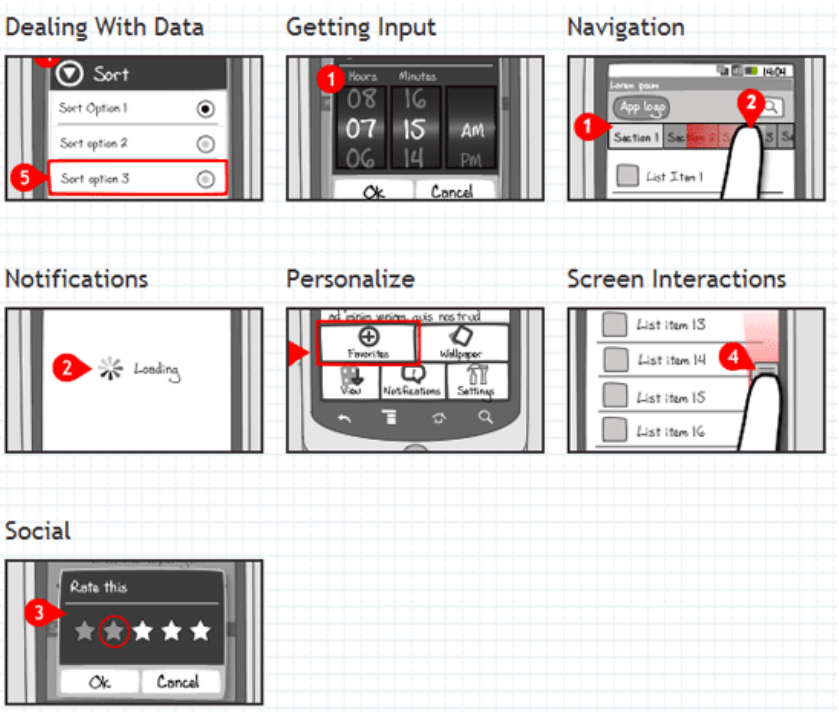

Figure 25. Android Patterns [35]

- **Dealing with Data:** Main characteristics of the patterns related to showing, watching, ordering, filtering, navigating and searching in a data set. Some examples would be the searching applications, state bars…

- **Getting Input:** It covers the common way that the users have to insert data in a specific context. For instance, when the user interacts with the different fields of a form (text boxes, spinners, check boxes…)

- **Navigation:** Navigation is the process of going from one place to another. It has two goals; firstly, navigation lets the user know which information is available on the

application and secondly; it helps the users to find the needed information the fastest the better. Here we will find examples of how to categorize and show the information on screen, use of contextual menus, toolbars…

- **Notifications:** Notifications alert the user about an application event. Maybe some of them will need a user answer or sometimes it will only give information. In this section we find how to use toast messages and progress bars.

- **Personalize:** By personalizing, users can change and save personal information and administrate security changes in the Android device. Here we find the patterns to do login screens, create shopping…

- **Screen Interactions:** Screen interactions include all the interactions that the user can do with his/her finger on the screen with one exception, the single tap, which always corresponds to de selection gesture.

- **Social:** The design patterns contained by this section allow the user to communicate and interact with other users. For instance, answering to a certain content through a comment, rating contents…

From the patterns, the best option to create the application menus has been thought. In the following sections the main screens design will be shown, like the initial (which is shown once the application is opened), the game one and the Wi-Fi-Direct one.

6.3. Initial screen

In the initial screen there must be shown the main actions that the user can do inside the application. Following the states transition diagram, a design of the initial screen has been done. In Android, there can be several types of resources designed and include them in an application. To achieve this, a XML file must be created with the menu definition in order to reference them with Java code afterwards. Handling the menus this way is great practice as it lets the menu contents to be separated of our application code. This way it is also easier to visualize the structure and content of the menu in a XML format:

menu.xml | strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/nueva_nota"
          android:icon="@drawable/escribir"
          android:title="@string/nuevo" />
    <item android:id="@+id/herramientas"
          android:icon="@drawable/opciones"
          android:title="@string/herramientas" />
</menu>
```

Figure 26. Menu code fragment

As a disadvantage in all the resource type added to our Android project, we need to convert them into manipulable programmable objects in order to be used. Below there is the needed code for using a menu resource in an application activity menu to deploy the menu options:

```
@Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.menu, menu);
        return true;
    }
```

Figure 27. Menu creation code fragment

The method used in onCreateOptionsMenu() that will allow us to use the MenuInflater class which is used to instant a menu XML resourceinside an object of the menu class. This is achieved by calling the inflate() method that receives as a first parameter the menu XML resource that will be assigned to the menu object that receives the main method onCreateOptionsMenu(). It must be said that this method will have to be called to the onCreate() method of the activity. And the same way, it will have to be included the corresponding import for the code to work with no problems. The initial screen final result is the following:

Thanks to this design can keep the same format as the original game, showing the face of the traditional user interface. In addition to the small amount of available options enables interactivity and ease of application for everyone.

This screen is the first screen you will see once you start any user application. For this reason it has been designed so that at a glance you can see the different game options that allows application.

The following sections show the layout of the main screen where you can see how keeping the same design throughout the application.

Figure 28. Initial screen

6.4. Game screen

When talking about the game screen, this will be achieved by code. This means that it will not use bitmaps when showing the user rackets or ball movements in the screen. As the application is different because of the fact of being a game application, Android provides APIS graphic series with a wide function range to manipulate images in bits map, 2D graphic, vector graphic tools, Drawable animation objects that supports several formats as jpg, png or gif, text creation and 3D graphics. As a graphic elaborator resource, Android provides Canvas, Paint, Path and Drawable classes.

Canvas class contains all the logic summarized in methods in order to draw. All the logic of how to draw circles, lines or rectangles is stored in the Canvas class. When we draw, we are in fact editing the pixel of an underlying bits map attached to Canvas itself. This bits map is always generated automatically when the views common onDraw() event is overcharged. What the bits map do is to save all the information of what is being drawn through Canvas methods.

Paint will offer the possibility of giving style and format giving the resources to the elements being drawn through some methods that we will see later on. We can also create our own bits map sketch in order to convert it as a parameter into the builder of a new Canvas as it is shown in the following g picture.

```
1 Bitmap bitmap = Bitmap.createBitmap(100, 100, Bitmap.Config.ARGB_8888);
2 Canvas c = new Canvas(bitmap);
```

Figura 29. Fragment de codi *canvas*

Every created class inherited from Views has an available onDraw to draw graphics and items part of a view drawing. Every View inherited object has with itself a ViewHolder, possessor of the views surface, in charge of facilitating the access to the Canvas drawing by its blockade for being then drawn with onDraw method and then, unblocking it to show it. This process done by holder is automatically done in Views objects so we cannot see what happens. It is executed in a non-visible code and we can only draw using onDraw to edit the methods possessed by Canvas.

A second important process is that in the case of overcharging onDraw, automatically there will be created a parameter as a Canvas object, representing the own Canvas personalized view and implicitly its bits map so that we can start drawing pixel. It will by default have a white background with the width and high of its container view. The following code shows an

example of a class that is directly a View class inherited where we overcharge its onDraw event.

```
public class Lienzo extends View {

         //constructor
        public Lienzo(Context context) {
            super(context);
        }
                //evento onDraw()
        @Override
        protected void onDraw(Canvas canvas) {
            // TODO Auto-generated method stub
            super.onDraw(canvas);
                        //aqui llamamos a Canvas ,sus métodos...
                                                }
}
```

Figura 29. Exemples d'ús del *canvas*

During the onDraw() event calling we will do all the needed modifications using Canvas inherent methods for modifying and adding graphics. Some of the ones used in the application are the following:

```
//dibujar arcos
canvas.drawArc();
//dibujar Texto
canvas.drawText();
//dibujar puntos
canvas.drawPoint/s();
//color de fondo del canvas
canvas.drawColor(Color.RED);
//dibujar un un Bitmap
canvas.drawBitmap()
//dibujar circulo
canvas.drawCircle()
//dibujar recta
canvas.drawRect()
```

Figura 30. Exemples d'ús del *canvas*

As it has been mentioned, it exists a Paint class which is called by Canvas in many of its methods as a parameter. It basically establishes the graphic applied styles that we draw in Canvas. Paint possesses several methods. The ones used in the application are the following:

```
//instanciamos un objeto de la clase paint
Paint pincel=new Paint();
//color...
pincel.setColor(Color.RED);
//ancho trazo...
pincel.setStrokeWidth(20);
//ancho texto...
pincel.setTextSize(textSize)
//color por Rgb...
pincel.setARGB(a, r, g, b);
```

Figura 31. Exemples d'ús del *paint*

After showing how to use Canvas and Paint, the result obtained for the application will be shown. As it can be observed, there has been crated a black background to simulate the original game and, using Canvas, the different objects have been drawn. The chosen colors are blue and red in order to differentiate both players (instead of the Pong original white). Finally, Instead of following more modern versions and getting the player to win at the best of five

games, I have created a score at the upper and lower side where there are the lives left. The player who first loses all the lives is the loser. The result is the following:

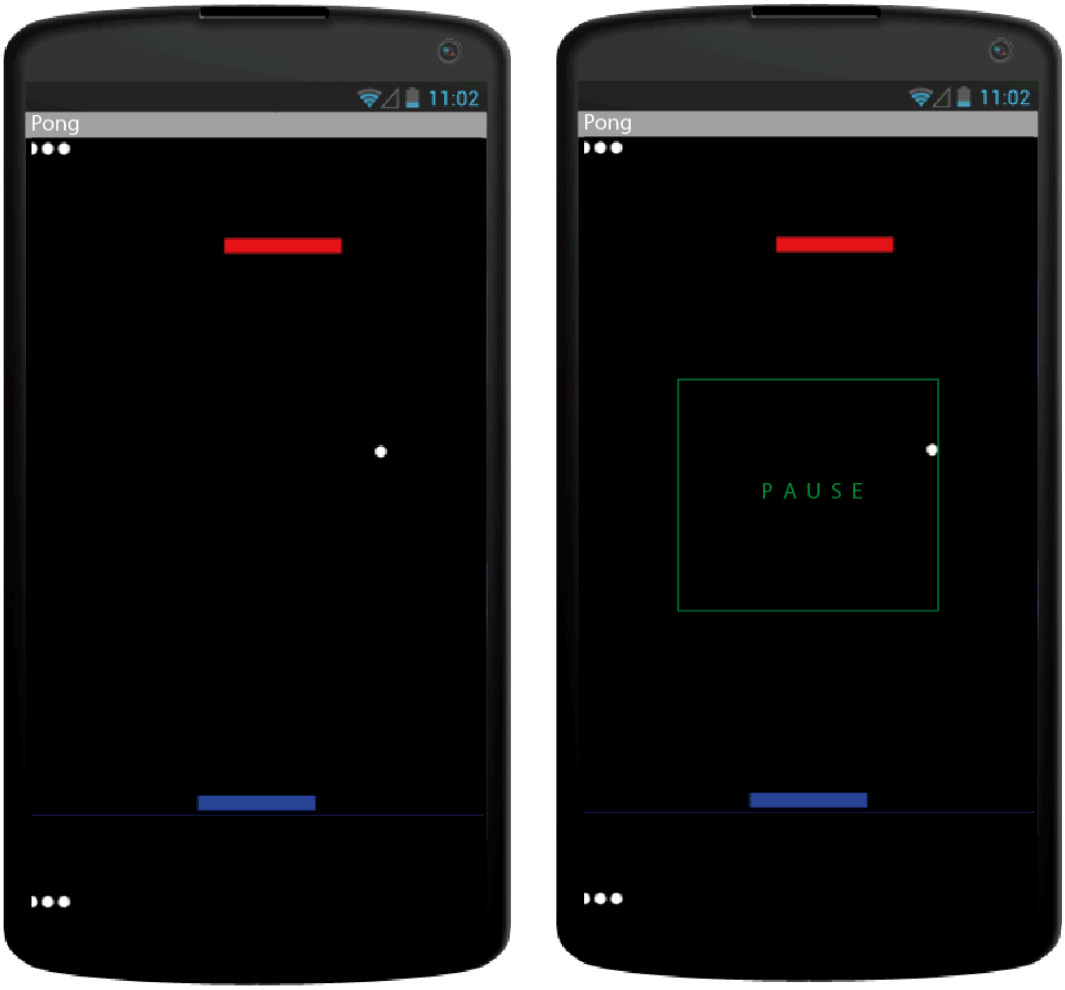

Figure 32. Game screens

You can see how the interface is faithful to the original except for the colors (the original was in black and white). In addition, to facilitate the game the user will always have control of the racket lower because this way you get a better view and control the game in general. Finally we should call that added functionality to pause the game. Thanks to this feature you can always started games continue from the same point where you stopped playing last time.

6.5. *Wifi Direct* connection screen

Another of the main screens in the application is the one that shows how to use Wi-Fi-Direct connection in order to start the game. In this case, there are two screens, one for the player who creates the game and another one for the player who wants to go in a created game. As it has been explained in previous sections, it is necessary that one of the devices create an access point in order that the other devices can connect to it.

The first screen that will be shown will be the new game creation one. In this case it is an easy form following the GettingInput pattern. It is only necessary the user to introduce the name of the game and it will be the name shown to the other users that want to connect to this game. The user has to also introduce a user name that the opponent will be able to see. Once this form is finished, it is necessary to tap the *New Game* button in order to start the game. Once the user taps the button, the device creates a Wi-Fi-Direct access point (in the implementing sections it will be explained how to do so) and only an opponent is needed. The final Wi-Fi-Connection screen is this:

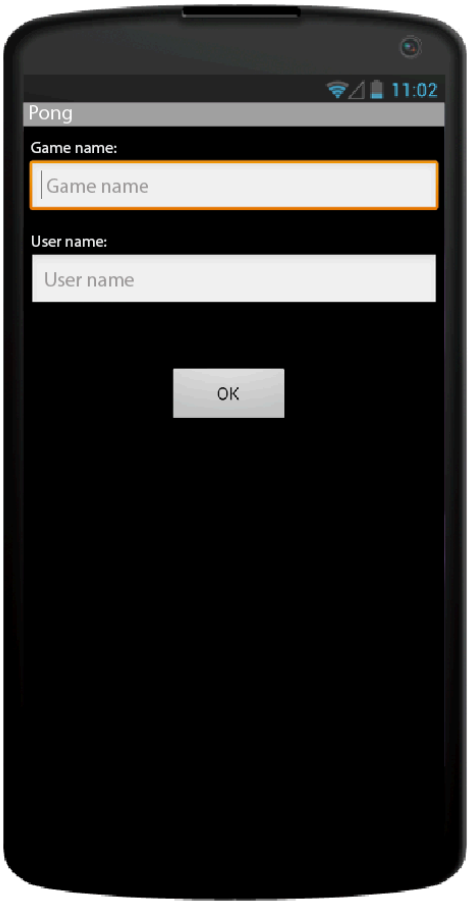

Figure 33. Create new game screen

On the other hand, it exists a screen needed for the opponent to connect into the created game. In this case, there is only a text space where the user has to insert the user name. In order to keep using the before mentioned patterns it has been used the Notifications pattern. In this case, only a waiting wheel is needed for the user to be certain that the device is searching for a game. Once a game is located, there is a games list shown to the user and the user has to choose a game. Once the user has done the chose, the game is started and both users connect with Wi-Fi-Direct. Here there is the waiting screen design:

Figure 34. Play into existing game

7. Implementation

7.1. Application development

After finishing the application design, it is time to develop it. For this reason this section focuses on the main characteristic of the application implementation as well as the most characteristic points of it. In the first section there will be shown the game implementation and, in the second, the Wi-Fi-Direct implementation.

7.1.1. *Pong*

7.1.1.1. *Handler*

In a videogame it is important to continually upgrade the game screen in order to always see the last position of the different screen components. For this reason, the main game loop usually runs in a hidden thread (callback). However, if the screen is tried to be upgraded, you will see Android gives an exception like this back:

```
android.view.ViewRoot$CalledFromWrongThreadException:
Only the original thread that created a view hierarchy can touch its views.
```

Figure 35. Error before use Handler

When anyone initiates a program in Android, the execution that is used is UIThread, which is the main conductor of the applications and the one in charge of upgrating the user interface. Why not to upgrade the screen game with UIThread? The logic of only one game is a work with a lot of data and it will make the application turn slower. What is more, from the 4.0 version, the own system prevents you from doing it. The only solution is to attach a thread to the game logic and paint all the screen components again. For this reason, in this project I will use Handlers (*API android.os.Handler*[1]).

Handlers implement a messages step mechanism between threads and they allow us to send messages from our wire to the UIThread. For this reason the Handler type method sendMessage is used in order to warn UIThread that it has to repaint all the screen items. In this project, the use of Handler it will be used for upgrading the screen each time a user modifies a game component. The following code will show how to upgrade the screen in case of modifications:

[1] http://developer.android.com/reference/android/os/Handler.html

```
class RefreshHandler extends Handler {
    @Override
    public void handleMessage(Message msg) {
        PongView.this.update();
        PongView.this.invalidate(); // Mark the view as 'dirty'
    }

    public void sleep(long delay) {
        this.removeMessages(0);
        this.sendMessageDelayed(obtainMessage(0), delay);
    }
}
```

Figure 36. Handler fragment to update the screen

Here there is a loop that upgrades the game in order to always see the las state and not to show the screen in items' parts like, for instance, the ball's progress:

```
private void notifyAllClients(MessageType type, String message) throws IOException {
    for(int i = 0; i < numberOfPlayers; i++) {
        DataOutputStream playersOutputStream = players[i].getOutputStream();
        Communicator.sendMessage(playersOutputStream, type, message);
    }
}
```

Figure 37. Handler fragment to notify

And here is the loop that performs the update the game to always see the latest status and display on the screen of the elements, such as the progress of the ball:

```
public void update() {
    if(getHeight() == 0 || getWidth() == 0) {
        mRedrawHandler.sleep(1000 / FPS);
        return;
    }

    if(!mInitialized) {
        initializePongView();
        mInitialized = true;
    }

    long now = System.currentTimeMillis();
    if(gameRunning() && mCurrentState != State.Stopped) {
        if(now - mLastFrame >= 1000 / FPS) {
            if(mNewRound) {
                nextRound();
                mNewRound = false;
            }
            doGameLogic();
        }
    }

    // We will take this much time off of the next update() call to normalize for
    // CPU time used updating the game state.

    if(mContinue) {
        long diff = System.currentTimeMillis() - now;
        mRedrawHandler.sleep(Math.max(0, (1000 / FPS) - diff) );
    }
}
```

Figure 38. Handler fragment to update

7.1.1.2. Artificial intelligence

In this application there is the possibility to directly play against the device. For this reason, some artificial intelligence characteristics have been implemented so that the user can play against the device as if it was a real user.

It is for this reason that, following some calculations, there has been some system artificial intelligence implemented. Firstly, an artificial intelligence that always followed the ball was invented. This means that the machine was usually the winner. Then, I wanted to develop a less perfect machine so that the user could also win. The idea was to create different functions and alternate them. The following code shows the swift conducting this decision:

```
private void doAI(Paddle cpu, Paddle opponent) {
    switch(mAiStrategy) {
    case 2: aiFollow(cpu); break;
    case 1: aiExact(cpu); break;
    default: aiPrediction(cpu,opponent); break;
    }
}
```

Figure 39. Switch fragment

Switch[2] is characterized by being a random number, depending on the grade of difficulty chosen by the user as it will be shown in the next section. In grade 1, it is perfect as aiExact() function calculates exactly where the ball will go and places the racket into the correct place:

```
private void aiExact(Paddle cpu) {
    cpu.destination = (int) mBall.x;
    cpu.setPosition(cpu.destination);
}
```

Figure 40. Fragment IA exact code

In the second case, aiFollow() follows the ball on screen and the opponent user sees how the racket moves to the place where the ball will go:

```
private void aiFollow(Paddle cpu) {
    cpu.destination = (int) mBall.x;
    cpu.move(true);
}
```

Figure 41. Fragment following ball IA code

Finally, there is a more extent function that predicts the ball location. This prediction is not exact which facilitates the opponent to win. Mainly it is characterized by observing the actual ball location, which means seeing if it bounces the walls, the distance between the rackets, of the actual racket location. The following code fragment shows some of the previously cited

[2] http://docs.oracle.com/javase/tutorial/java/nutsandbolts/switch.html

characteristics and make possible that the user can have the possibility to win against the machine:

```
private void aiPrediction(Paddle cpu, Paddle opponent) {
    Ball ball = new Ball(mBall);

    // Special case: move torward the center if the ball is blinking
    if(mBall.serving()) {
        cpu.destination = getWidth() / 2;
        cpu.move(true);
        return;
    }

    // Something is wrong if vy = 0.. let's wait until things fix themselves
    if(ball.vy == 0) return;

    // Y-Distance from ball to Rect 'cpu'
    float cpuDist = Math.abs(ball.y - cpu.centerY());
    // Y-Distance to opponent.
    float oppDist = Math.abs( ball.y - opponent.centerY() );

    // Distance between two paddles.
    float paddleDistance = Math.abs(cpu.centerY() - opponent.centerY());

    // Is the ball coming at us?
    boolean coming = (cpu.centerY() < ball.y && ball.vy < 0)
        || (cpu.centerY() > ball.y && ball.vy > 0);

    // Total amount of x-distance the ball covers
    float total = ((((coming) ? cpuDist : oppDist + paddleDistance)) / Math.abs(ball.vy)) * Math.abs( ball.vx );

    // Playable width of the stage
    float playWidth = getWidth() - 2 * Ball.RADIUS;


    float wallDist = (ball.goingLeft()) ? ball.x - Ball.RADIUS : playWidth - ball.x + Ball.RADIUS;

    // Effective x-translation left over after first bounce
    float remains = (total - wallDist) % playWidth;
```

Figure 42. Fragment IA code

Thanks to this code it is possible for the user to live a better interaction with the machine because the application can predict where the ball will go the same way as a real user. This type of prediction makes possible a more real interaction between the user and the machine.

7.1.1.3. *Shared Preferences*

Preferences are is data that an application has to safe in order to personalize the user experience like personal information, presentation options… The application preferences can be stored in a database but Android provides an alternative method specifically designed for this type of data: *shared preferences*[3]. Each preference will be stored in a key-value way which means that each of them will be composed by an only identifier (like an email)) and a value associated to the identifier (for instance pong@@pong.cat). Furthermore and unlike a database, data is not saved in a binary database file but in XML file. A file is an XML document with all the information in the most structured abstract and reusable as possible. Such files are characterized by being easily understood by people such as computers. In addition to clearly separate the information it contains and its syntactic analysis is through design guidelines that need to follow. Finally we should have called a hierarchical structure and is designed for any language or alphabet.

The API for the handling of these preferences is really easy. Every management is centered in the SharedPreferences class which will represent a preference collection. An Android application can manage several preferences collections that will be different by a unique identifier. In order to get a reference of a specific collection it is necessary to use the getStartedPreferences() method in which we will pass the collection identifier and the access way. The access way will tell us which applications will have access to the preferences collection. So, we have three main possibilities:

- **MODE_PRIVATE:** Only our application has access to these preferences.
- **MODE_WORLD_READABLE:** All the applications can read these preferences, but only ours can modify them.
- **MODE_WORLD_WRITABLE:** All the applications can read and modify these preferences.

The last two options are relatively dangerous as any application could modify our preferences. In fact, they have been declared obsolete in API 17 (Android 4.2) deleting this functionality.

It is necessary to first obtain a reference to a SharedPreferences variant and once it is obtained, we can obtain, insert and modify preferences using get or put methods attached to

[3] http://developer.android.com/reference/android/content/SharedPreferences.html

the data type of each preference. The following code shows how to store the user preferences like the ball speed or player lives:

```
protected void loadPreferences(SharedPreferences prefs) {
    Context ctx = getContext();
    Resources r = ctx.getResources();

    mBallSpeedModifier = Math.max(0, prefs.getInt(Pong.PREF_BALL_SPEED, 0));
    mMuted = prefs.getBoolean(Pong.PREF_MUTED, mMuted);
    mLivesModifier = Math.max(0, prefs.getInt(Pong.PREF_LIVES, 2));
    mCpuHandicap = Math.max(0, Math.min(PLAYER_PADDLE_SPEED-1, prefs.getInt(Pong.PREF_HANDICAP, 4)));

    String strategy = prefs.getString(Pong.PREF_STRATEGY, null);
    String strategies[] = r.getStringArray(R.array.values_ai_strategies);

    mAiStrategy = 0;
    // Linear-search the array for the appropriate strategy index =/
    for(int i = 0; strategy != null && strategy.length() > 0 && i < strategies.length; i++) {
        if(strategy.equals(strategies[i])) {
            mAiStrategy = i;
            break;
        }
    }
}
```

Figure 43. Fragment user preferences

To upgrade or insert new preferences, the process will be that easy too with only one difference; the update or insertion will not be directly done over the SharedPreferences object but over its SharedPreferences.editor edition object. To this last object we will access with an edit() method of the SharedPreferences type. Once the editor reference is obtained, we will use put methods corresponding to the data type of each preference to update or insert its value like, for instance, putString(key, value) to update a String type preference. Similarly to the already seen get methods, there will be available put methods for all type of basic data: putInt(), putFloat(), putBoolean()…

Finally, once all the needed data is updated/inserted we will call the commit() method to confirm the changes. The following code shows the function that allows to disable the sound in the game:

```
public void setMuted(boolean b) {
    // Set the in-memory flag
    mMuted = b;

    // Grab a preference editor
    Context ctx = this.getContext();
    SharedPreferences settings = PreferenceManager.getDefaultSharedPreferences(ctx);
    SharedPreferences.Editor editor = settings.edit();

    // Save the value
    editor.putBoolean(Pong.PREF_MUTED, b);
    editor.commit();

    // Output a toast to the user
    int rid = (mMuted) ? R.string.sound_disabled : R.string.sound_enabled;
    Toast.makeText(ctx, rid, Toast.LENGTH_SHORT).show();
}
```

Figure 44. Fragment mute game

However, as it has been mentioned before, the user characteristic are not stored in a database but in a XML file. It is for this reason that, inside the XML folder of our project, there will be created a new XML document with the user stored information. The following code shows part of the user stores preferences:

```
<PreferenceCategory android:title="@string/label_gameplay">
    <ListPreference
        android:key="strategy"
        android:title="@string/label_ai_strategy"
        android:summary="@string/summary_ai_strategy"
        android:entries="@array/labels_ai_strategies"
        android:entryValues="@array/values_ai_strategies" />
```

Figure 45. Fragment XML code

Thanks to this feature it is possible to store all the user preferences and avoid restore the application without considering these preferences. This feature, although secondary, is very important in applications that want to use every day as more and preferences defined by the user can be a heavy work to the point that the user removes your application the device.

One of the main reasons that has been used SharedPreferences is its ease of development and offers great functionality within an application. Mainly it is important that users get a personalized navigation within any application, so always keep stored preferences enables a player that does not need to change every time you enter the application preferences play.

7.1.1.4. *PreferenceActivity*[4]

The previous section was about Shared Preferences, a mechanism that allows us to easily manage an application options letting us to save the in XML in a transparent way for the programmer. By now, we only know how to use them with code. However, Android offers an alternatively way to define using XML a set of options for an application and to create the needed screens to let the user modify them as he/she wants.

As we have indicated, we will define our screen options by an XML in a similar way of how we define any layout, even though in this case we will have to include it in an XML.

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android"
    android:key="pong_main">
    <PreferenceCategory android:title="@string/label_sound">
        <CheckBoxPreference
            android:key="muted"
            android:title="@string/label_muted"
            />
    </PreferenceCategory>
    <PreferenceCategory android:title="@string/label_gameplay">
        <ListPreference
            android:key="strategy"
            android:title="@string/label_ai_strategy"
            android:summary="@string/summary_ai_strategy"
            android:entries="@array/labels_ai_strategies"
            android:entryValues="@array/values_ai_strategies" />

        <com.hlidskialf.android.preference.SeekBarPreference
            android:key="handicap" android:title="@string/label_cpu_handicap"
            android:text="-%s" android:summary="@string/summary_cpu_handicap"
            android:dialogMessage="@string/label_cpu_handicap"
            android:defaultValue="0" android:max="7" />

        <com.hlidskialf.android.preference.SeekBarPreference
            android:key="ball_speed" android:title="@string/label_ball_speed"
            android:text="+%s" android:summary="@string/summary_ball_speed"
            android:dialogMessage="@string/label_ball_speed"
            android:defaultValue="0" android:max="15" />

        <com.hlidskialf.android.preference.SeekBarPreference
            android:key="lives" android:title="@string/label_lives"
            android:text="+%s" android:summary="@string/summary_lives"
            android:dialogMessage="@string/label_lives"
            android:defaultValue="0" android:max="9" />
    </PreferenceCategory>
</PreferenceScreen>
```

Figure 46. Fragment XML PreferenceActivity code

The main container of our preference screen will be the <PreferenceScreen> item. This item will represent the options screen itself, in which there will be the rest of items included. In it, we will be able to include our option list organized by sections that will be

[4] http://developer.android.com/reference/android/preference/PreferenceActivity.html

represented with the <PreferenceCategory> item which we will give a descriptive text using its attribute android:tittle. In each section we will be able to add any amount of otions, which can be of different types. Here there are the main types:

- **CheckBoxPreference:** Selectable brand
- **EditTextPreference:** Simple text chain
- **ListPreference:** Selectable valuable list (exclusive)
- **MultiSelectListPreference:** Selectable valuable list (inclusive)

Depending on what is needed to do, the different type of them can be used. In this case, the shown code use will be necessary as small user preferences must be stored like the sound, ball speed…

7.1.2. *Wifi Direct*

7.1.2.1. *BroadcastReciver*

A broadcast receiver (*BroadcastReceiver*[5] *en l'API 19*) receives and reacts to broadcast global warnings. It exists a lot of them originated by the system; as low power, incoming call… However, applications can also send broadcasts. Broadcast receivers do not have user interface but can start an activity or create a notification to inform the user.

The cycle of life of a Broadcast Receiver is very simple as it only has an onReceive() method. In fact, a Broadcast Receiver object only exists during the call to onReeceive(). The system creates the BroadcastReceiver which calls this method and once it has finished, it destroys the object. An important detail is that it is not necessary to run the application to activate it.

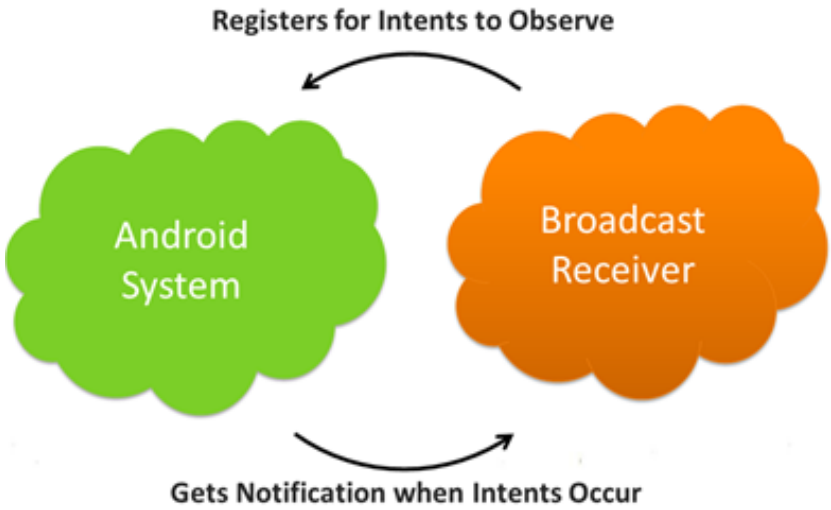

Figure 47. BroadcastReceiver schema [36]

The onReceive() method is executed through the main application wire. This is why it cannot blockthe system (see the activity cycle of life previously touched). If you have to do an action that can block the system, it will be necessary to throw a secondary wire. If we want a time persistent action it will be necessary to throw a service. From a Broadcast receiver you cannot show a dialog box or joining a service (bindService()), but instead of it you can show a notification. What is more, startService() can be used to start a service in the device itself. An application can register a broadcast receiver in two different ways: in AndroidManifest.xml and in the execution time by registerReceiver() method

[5] http://developer.android.com/reference/android/content/BroadcastReceiver.html

This means that our Broadcast Receiver will configure at a determinate system action, and the own system will be the one in charge of telling the Broadcast Receiver that this action has happened. This way, we would be able to create, for instance, a Broadcast receiver that was noticed when the system would have completed the starting process, registering it to ACTION_BOOT_COMPLETED.

In this project we will have two types of Broadcast receiver; a first one to communicate every time that there is a change in the game (ball movement, racket movement…) and another one to check if active games exist or not. The first case is shown in the following code and it is based in paying attention in order to know the game changes:

```
public class GameStateChangeReceiver extends BroadcastReceiver {
    private static final String TAG = "GAME_STATE_CHANGE_RECEIVER";
    private GameBoard gameBoard;
    private BoardView boardView;
    private String currentUserName;

    public GameStateChangeReceiver(GameBoard gameBoard, BoardView boardView) {
        this.gameBoard = gameBoard;
        this.boardView = boardView;
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        MessageType type = MessageType.valueOf(intent.getIntExtra(Constants.MESSAGE_TYPE, -1));
        String message = intent.getStringExtra(Constants.MESSAGE);
        String userName = intent.getStringExtra(Constants.USER_NAME);

        Log.d(TAG, "Received message: " + type + ", " + message);
        type.processMessage(this, userName, message);
    }
```

Figure 48. Fragment BroadcastReceiver code

In the second case, it is a code that will be necessary to automatically receive the available games without the need for the user of updating the screen. As it can be observed in the following code, more than a Broadcast receiver uses the Wifi manager, a type of class that will be explained in the next section.

```
public abstract class WifiReceiver extends BroadcastReceiver {
    private WifiManager wifiManager;
    private Map<String, String> foundGames;

    public WifiReceiver(WifiManager wifiManager) {
        this.wifiManager = wifiManager;
        foundGames = new HashMap<String, String>();
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        List<ScanResult> resultList = wifiManager.getScanResults();
        String gameNamePrepend = "\"" + GameWiFiManager.GAME_TAG;
        for(ScanResult result : resultList) {
            if(result.SSID.startsWith(gameNamePrepend)) {
                String name = result.SSID.replace(gameNamePrepend, "").replace("\"", "");
                foundGames.put(name, result.SSID);
            }
        }

        if(!foundGames.isEmpty()) {
            processFoundGames();
        }
    }

    protected abstract void processFoundGames();

    public Map<String, String> getFoundGames() {
        return foundGames;
    }
}
```

Figure 49. Fragment BroadcastReceiver code

7.1.2.2. *WifiManager*

This class provides the main API to manage all the aspects of the Wi-Fi connection. With this API several actions inside an application can be done. The main ones are:

- To list the configured networks as well as update and view all the network attributes.
- To able and disable the existent Wi-Fi network device and know all dynamic the information about the network state in which is connected.
- To explore and decide which access point to connect to from all the access points exploration.
- To have actions in each case or possible situation depending on the Wi-Fi state.

This is the API needed for doing specific Wi-Fi operations as, if we do the operations needed referred to an abstract level of network connections, *ConnectivityManager*[6] is used. In the following code, used in the application, it is possible to observe how, together with Broadcast receiver, it is possible to implement an abstract class that can be aware of the Wi-Fi

[6] http://developer.android.com/reference/android/net/ConnectivityManager.html

networking news. In this case, the function getScanResults() have been used and it is characterized by showing all the available access point results:

```
public abstract class WifiReceiver extends BroadcastReceiver {
    private WifiManager wifiManager;
    private Map<String, String> foundGames;

    public WifiReceiver(WifiManager wifiManager) {
        this.wifiManager = wifiManager;
        foundGames = new HashMap<String, String>();
    }
```

Figure 50. Fragment WifiReceiver code

In this class you can see how abstract WifiReceiver used to store a HashMap different game rooms. In other words, information stored on different access points close to the device on which the application is running. The following code shows the waiting message received from a BroadcastReceiver, ie, the function is waiting to receive a message broadcast application. Upon receipt of this message obtained different game rooms (access points), which insert a list later to show it to the user.

```
    @Override
    public void onReceive(Context context, Intent intent) {
        List<ScanResult> resultList = wifiManager.getScanResults();
        String gameNamePrepend = "\"" + GameWiFiManager.GAME_TAG;
        for(ScanResult result : resultList) {
            if(result.SSID.startsWith(gameNamePrepend)) {
                String name = result.SSID.replace(gameNamePrepend, "").replace("\"", "");
                foundGames.put(name, result.SSID);
            }
        }

        if(!foundGames.isEmpty()) {
            processFoundGames();
        }
    }

    protected abstract void processFoundGames();

    public Map<String, String> getFoundGames() {
        return foundGames;
    }
}
```

Figura 51. Fragment de codi *WifiReceiver*

Thanks to this set of code and API WifiReceiver is possible to display on screen the user information for access points created by nearby devices. This function will run until the user select and start a game on one of the access points. If the user does not select any, the application will continue listening and showing all access points obtained by internal messages broadcast.

7.1.2.3. Interface *Comparable*

Mainly needed a class comparing the current situation with the player's racket final status (to move the racket smoothly over the screen). There were various options such as comparator and comparable interfaces to perform this function. But one of the main concerns was the differences between comparator and comparable or ordered collection of objects if their ID or by name. The interface is characterized Comparable to classify objects but must be implemented by objects to classify. It is for this reason that to use interface, you must implement the method compareTo (Object). This interface must be implemented in all classes that want to define a natural order of their instances. In other words, we should be implemented whenever you need the functionality of the interface in a class of our own.

The order rule has to be transitive (if x.compareTo(i)<0 and y.compareTo(z)<0, then x.compareTo(z)<0) and invertible (the result sign of x.compareTo(i) has to be the same to the denialof the result sign of y.compareTo(x) for all the X and Y combinations). What is more, it is advised (but not compulsory) to do so and, only if the result of x.compareTo(i) is zero, the x.equals(i) result is right.

After seeing the rule necessary to compare and compare comparable interface was selected second by its ease of use. To see an example and improve your understanding of this interface then displays the details of code which makes use of this class in the project:

```
public class PlayerDetails implements Comparable<PlayerDetails> {
    private String name;
    private int currentPosition;
    private int result;
    private DataInputStream inputStream;
    private DataOutputStream outputStream;

    @Override
    public int compareTo(PlayerDetails playerDetails) {
        if(result > playerDetails.getResult()) {
            return 1;
        }

        if(result < playerDetails.getResult()) {
            return -1;
        }

        if(currentPosition > playerDetails.getCurrentPosition()) {
            return 1;
        }

        if(currentPosition < playerDetails.getCurrentPosition()) {
            return -1;
        }

        return 0;
    }
```

Figure 52. Fragment Comparable code

Mainly we used the interface in order to compare comparable without using generic methods that could produce errors or problems when checking the current situation and the future. For this reason it was decided to develop a function able to set a number of parameters to compare these data and thus ensure that the operation is correct.

7.1.2.4. *TimerTask*

It exists a problem with the graphic GUI. Several times we need to update or interact with a graphic interface from a thread that is not the main one. Precisely, in this case, a periodic task (thread) is needed to show a pop-up in the interface (activity) informing that it is waiting for a canvas update. As this method would be annoying for the user (each time that the ball moves there would be necessary to inform the user) and heavy for the application to send too many messages, we decided to make use of the class *TimerTask*[7].

The first important thing to know is how to periodically execute a method in Android. For this reason, we can use a Java native class called Timer Task. We can configure the delay (of the first execution) and the period (of the following executions). If from the run method of this Timer Task we try to modify anything related to the graphic interface that it is executed in the main thread, we will see how the application gets stack or shows an error warning that the interface can only be accessed from the main thread (activity).

Below there is the code of how a Timer Task has been created and configured for this project's application, as it will be used to refresh the game's view:

```
public class ViewRefresher extends TimerTask {

    private final View view;

    public ViewRefresher(View view) {
        this.view = view;
    }

    @Override
    public void run() {
        view.postInvalidate();
    }
}
```

Figure 53. Fragment TimerTask code

[7] http://developer.android.com/reference/java/util/TimerTask.html

As it can be observed, the code is shorts as we will only have the run function that will be executed every time that we want to refresh the view. In this case, inside the run function there is a call to the postInvalidate() function. Each class derived from View has the function of invalidate and posInvalidate method. Invalidate warns the system that the actual view has changed and that it has to be repainted as quickly as possible. As this method can only be called by UIThread, it is necessary to have another method in order to use it when it is not in the UIThread and it still wants to notify the system that the view has changed, The postInvalidate method notifies the system of an UIThread and the view in again drawn in the next event in the UIThread as fast as possible. Thanks to the union of these methods, it is possible to have a better interaction between the user and the videogame.

7.1.2.5. *Socket*

In order to connect two mobile phone devices with each other and be able to play, it is necessary to develop two classes: a client class and a server class. One of the devices will be the one to start the server class and automatically will convert into an access point. From this moment on, any device can connect to it.

The socket interface[8] defines the rules that a program has to follow in order to use the transport level services in a TCP/IP networking. A socket is the final point of a bidirectional communication between two programs that interchange information through the Internet. As in a same device or computer it is possible to simultaneously execute different applications that use the Internet to communicate, it is important to identify each socket in a different direction. A socket is usually identified by the IP address of the device plus a port number (of 16 bytes).

It is necessary to firstly see the operation of a client-server to develop the needed classes for its right operation. As it can be seen in the following diagram, the connection is firstly established, then the server waits for the clients to connect and one connected, a request is done and a reply is received until the connection is closed. We will try to do the same.

[8] http://developer.android.com/reference/java/net/Socket.html

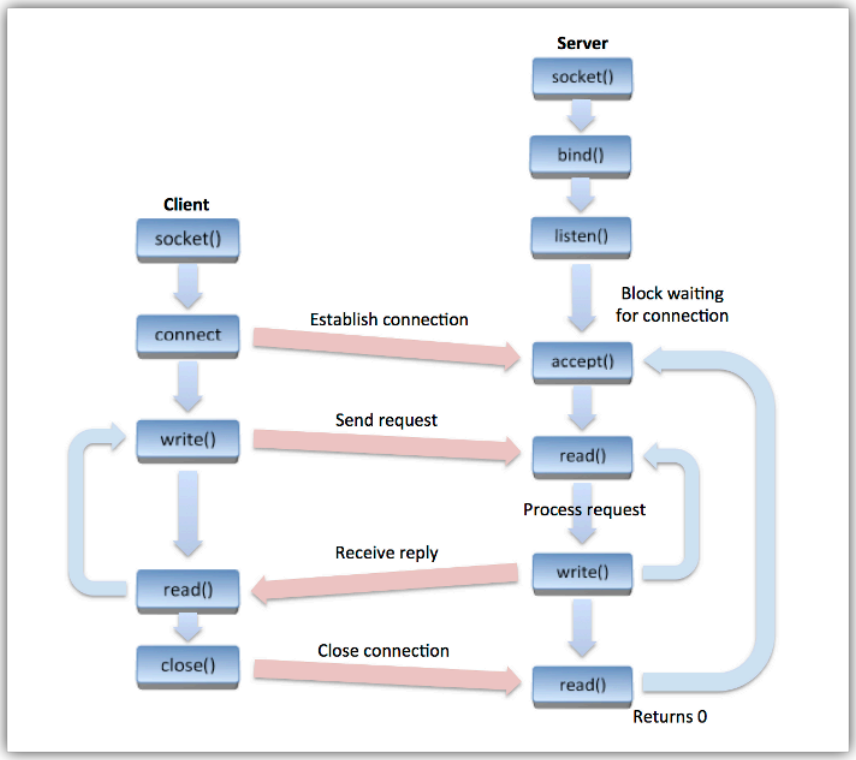

Figure 54. Sockets connection diagram

In Android, the socket management is like in Java (at least with the TCP). When we work with sockets, we can use it in clients and in the server. The I/S is done through InputStream and OutputStream objects associated to sockets. In the server it is necessary to do the following:

- Create the socket server
- Accept a client
- Obtain the clients' InputStream and/or OutputStream
- Create some InputStream and/or OutputStream more adequate to our needs
- Read and write the client data
- Close the socket

In the server, a serversocket is created that has the port number as a parameter. This number can be from 1 to 65535 and from 1 to 1023 are reserved for system services (SSH, SMTP, ftp, mail, www,telnet…) From 1024 on can be freely used. The server creates a socket server and has as a parameter the port in which the incoming petitions were being listened. Once the client is connected, we have to obtain its outputstream or inputstream with the getoutputStream() or getInputStream() methods. The outputStream is used for sending data and the InputStream for reading received data. We can also send and receive objects in our application using the message_data and obtaining this way a specific type of data.

In the following code fragments it is possible to see the connection stabished by a server and the client. In the first case, the server creates an access point visible for all the devices using Wi-Fi-Direct technology. Thanks to this server, the other users will be able to use the device as an access point in order to allow the incoming connections and be able to play.

```
@Override
public void run() {
    try {
        gameWiFiManager.initializeAccessPoint(gameName);
        gameWiFiManager.initializeServerSocket();
        waitForPlayers();
        runGame();
    } catch(GameEndedException gee) {
        Log.d(TAG, "game ended exception caught!! (server.run())");
    } catch (Exception e) {
        Log.d(TAG, "Error while starting game");
        try {
            notifyAllClients(MessageType.GAME_OVER, "");
        } catch(Exception ex) {
            //nothing can be done
        }
    } finally {
        try {
            gameWiFiManager.closeServerSocket();
        } catch(Exception ex) {
            //nothing can be done
        }
        gameWiFiManager.bringBackInitialApConfig();
        gameWiFiManager.bringBackInitialNetwork();
    }
}


private void waitForPlayers() throws IOException, GameEndedException {
    for(int i = 0; i < numberOfPlayers; i++) {
        Socket socket = gameWiFiManager.acceptConnection();
        socket.setSoTimeout(500);
        players[i] = new PlayerDetails();
        players[i].setInputStream(socket.getInputStream());
        players[i].setOutputStream(socket.getOutputStream());
        Log.d(TAG, "Connection from client accepted");
    }
}
```

Figure 55. Fragment sockets code

In the above code shows how to develop the method run on the server. This can be seen as gameWifiManager initialized by a new access point created from the initialization of a new socket. After the initialization, the device remains in the hope of connecting other devices. In this case, the method has been developed waitForPlayers () characterized to allow all incoming connection. If a player is properly connected to the new access point created, the application

will store all the data of the player. After confirming the correct socket connection between two devices, the game will start.

In this first version of the server functionality, as you can see, there is no security. That is, once created access point, it is completely free allowing the connection of any user. One of the objectives listed in the chapter on future work (Chapter 8.1.) Is the use of a password in order to connect two devices.

On the other hand, has developed the client. Below is a snippet of the method that the client will need to connect a device to the device currently acts as a server:

```
public class Client extends Thread {
    private static final String TAG = "CLIENT";
    private Context context;
    private DataInputStream inputStream;
    private DataOutputStream outputStream;
    private String name;
    private Socket socket;

    public Client(String name, Context context) {
        this.name = name;
        this.context = context;
    }

    @Override
    public void run() {
        try {
            init();
            sendMessageToServer(MessageType.NAME_INTRODUCTION, name);
            while(true) {
                handleServerMessage();
            }
        } catch (Exception e) {
            Log.d(TAG, "IO problem with sockets: " + e.getMessage());
            emitLocalStateChange(MessageType.GAME_OVER, "");
        } finally {
            try {
                socket.close();
            } catch (Exception e) {
                //nothing can be done
            }
        }
    }

    private void init() throws IOException {
        Log.d(TAG, "Connecting to server...");
        socket = new Socket(Constants.HOST, Constants.PORT);
        Log.d(TAG, "Server socket: " + socket);
        inputStream = new DataInputStream(socket.getInputStream());
        outputStream = new DataOutputStream(socket.getOutputStream());
        Log.d(TAG, "Connection to server established");
    }
```

Figure 56. Fragment client code

In this case, once selected access point in which you want to connect the main method run () function call init(). This function is characterized by trying to create a new socket that allows connection to the device has already prepared the access point. With this new device establishes a new socket connection to the server and to expedite the start of the game, once connected sockets, the information is sent as the username. As seen in the above code, the server will get the user name that will be necessary when storing the result of the game. Once the socket connection must be closed to terminate the connection.

8. Conclusions and future work

Finally a functional application has been achieved, both autonomously (wireless) and with the possibility of connecting to other devices. Even though the game is simple, it is characterized by being one of the first videogames to be developed in much more evolved technology. This has allowed to create a new application with different characteristics to the ones found in the market.

I have learnt about new and interesting technologies like Wi-Fi-Direct through this project, important for future creations as nowadays the interaction and connection between devices is an important tool to know. Some examples that are already in the market would be wireless printing machines or even televisions with this type of connectivity.

Thanks to the knowledge achieved through the creation of this project, I can now develop new applications and/or implement new functionalities to the existent ones. What is more, it has been a personal challenge as, although not all the initial objectives have been completed, the aim was for the project to be of personal growth and it is for this reason that I chose a non-well documented type of technology but very important nowadays.

On the other hand, finishing the project has shown me that, after having finished the four years that has lasted my degree at *Universitat de Lleida*, I can use all the knowledge learnt to develop my own applications. The fact of being able to use all this knowledge and the personal satisfaction of creating a project from the very beginning increases my motivations to keep developing and implementing the pieces of this project that were not finished because of lack of time.

8.1. Future work

During the duration of this project I have mentioned different possible improvements or new functionalities that could be taken into from this project on. It is for this reason that there has been made a summary of all the possible future developments of this project:

- Implementation of the failure part of the project because of time lack: smartphone devices connection server in order to play online.

- Develop inside the game a user and statistics system that allowed the user to create an account to modify personal data and see a list of all the punctuations achieved through all the games
- Modify the Wi-Fi-Direct part in order to use routers and existent access points and improve the Wi-Fi distance functionality.
- Enhance the game, making it public into the applications market and increase the functionality like getting customized rackets and balls, exclusive wallpapers…
- Create an online ranking with the possibility of creating an online championship with an own server.
- Using other type of technologies learnt in the Computer Engineering degree as Node or Google App Engine to create a backend or the own server.
- Develop the same application for IOS devices and this way having an application compatible with any type of smartphone.

Bibliography

Books

DARWIN, IAN F. (2012). *Android Cookbook* – O'Reilly Media

GARGENTA, MARCO & NAKAMURA, MASUMI (2014). *Learning Android, 2nd Edition* – O'Reilly Media

GRAMLICH, NICOLAS (2006). Andbook!

Web pages

[1] http://en.wikipedia.org/wiki/OXO

[2] http://www.ideafinder.com/history/inventions/pong.htm

[3] http://en.wikipedia.org/wiki/Computer_Space

[4] http://en.wikipedia.org/wiki/Video_game_genres

[5] https://www.open2study.com/courses/concepts-in-game-development

[6] http://ctb.ku.edu/en/table-of-contents/assessment/assessing-community-needs-and-resources/swot-analysis/main

[7] http://arstechnica.com/gadgets/2014/06/building-android-a-40000-word-history-of-googles-mobile-os/

[8] http://cv.uoc.edu/~javiercg/practica_final/modulo5_1.2.3.html

[9] http://www.androidcurso.com/index.php/tutoriales-android/37-unidad-6-multimedia-y-ciclo-de-vida/158-ciclo-de-vida-de-una-aplicacion

[10] http://en.wikipedia.org/wiki/Android_version_history

[11] http://www.wi-fi.org/discover-wi-fi/wi-fi-protected-setup

[12] http://www.wi-fi.org/discover-wi-fi/wi-fi-direct

[13] http://www.blogopeda.com/que-es-el-wifi-direct-y-como-funciona/

[14] https://www.wi-fi.org/certified-products-search

[15] http://www.aprenderaprogramar.com/index.php?option=com_content&id=392:la-maquina-virtual-java-jvm-o-java-virtual-machine-compilador-e-interprete-bytecode-cu00611b&Itemid=188

[16] http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-2.html

[17] http://source.android.com/devices/tech/dalvik/

[18] http://www.oracle.com/technetwork/developer-tools/eclipse/downloads/index.html?ssSourceSiteId=opn

[19] http://developer.android.com/guide/practices/screens_support.html

[20] http://www.androidpatterns.com/

[21] http://developer.android.com/reference/java/net/ServerSocket.html

[22] http://www.android-so.com/programar-en-android

[23] http://faqsandroid.com/actividades-y-layouts-en-android-capitulo-3-curso-android/

[24] http://www.androidexperto.com/aprender-android/versiones-android/

[25] http://www.omicrono.com/2011/11/la-tecnologia-wifi-direct-se-apunta-al-dlna/

[26] http://www.techradar.com/news/phone-and-communications/mobile-phones/wi-fi-direct-what-it-is-and-why-you-should-care-1065449

[27] http://www.techradar.com/news/phone-and-communications/mobile-phones/wi-fi-direct-what-it-is-and-why-you-should-care-1065449

[28] http://wiki.inf.utfsm.cl/index.php?title=M%C3%A1quinas_virtuales

[29] http://stackoverflow.com/questions/7541281/what-is-dalvik-and-dalvik-cache

[30] http://www.teehanlax.com/blog/density-converter/

[31] http://www.teehanlax.com/blog/density-converter/

[32] http://www.teehanlax.com/blog/density-converter/

[33] http://www.teehanlax.com/blog/density-converter/

[34] http://www.teehanlax.com/blog/density-converter/

[35] https://developer.android.com/design/patterns/index.html

[36] http://www.edureka.in/blog/what-is-android/

[37] http://www.blogopeda.com/comparte-archivos-por-wifi-direct/

[38] http://pcworld.com/article/208778/Wi_Fi_Direct_vs_Bluetooth_4_0_A_Battle_for_Supremacy.html

Other sources consulted:

ong Game [Consulted: February 15th 2014] Available on the internet:

http://www.ponggame.org/

Android developers [Consulted: February 25th 2014] Available on the internet:

http://developer.android.com/

Wikipedia [Consulted: March 3rd 2014] Available on the internet:

http://en.wikipedia.org/wiki/Pong

Torbjorn Kvale [Consulted: March 8th 2014] Available on the internet:

https://github.com/torbjokv/BrickBreaker

Java Code Geeks [Consulted: March 11th 2014] Available on the internet:

http://www.javacodegeeks.com/tutorials/android-tutorials/android-game-tutorials/

Kilobolt [Consulted: March 18th 2014] Available on the internet:
http://www.kilobolt.com/game-development-tutorial.html
Google developers [Consulted: March 20th 2014] Available on the internet:
https://developers.google.com/games/services/android/quickstart
Code project [Consulted: March 28th 2014] Available on the internet:
http://www.codeproject.com/Articles/188957/Simple-Android-Ball-Game